Rekomendasi Anime berdasarkan String Matching dengan Voice Recognition

Ilham Wahabi / 13516141
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13516141@std.stei.itb.ac.id
wahabiputra@gmail.com

Abstract— Mendapatkan rekomendasi anime dari data yang tersedia melalui API (Application Programming Interface) berdasarkan deskripsi yang pengguna berikan baik melalui teks maupun perintah suara bawaan perambah modern yang diproses menggunakan algoritma pencocokan string yakni Knuck-Morris-Pratt, Boyer-Moore, atau Regular Expression. Disajikan dalam bentuk situs web sehingga mudah diakses.

Kata kunci—web; anime; front end; voice recognition;

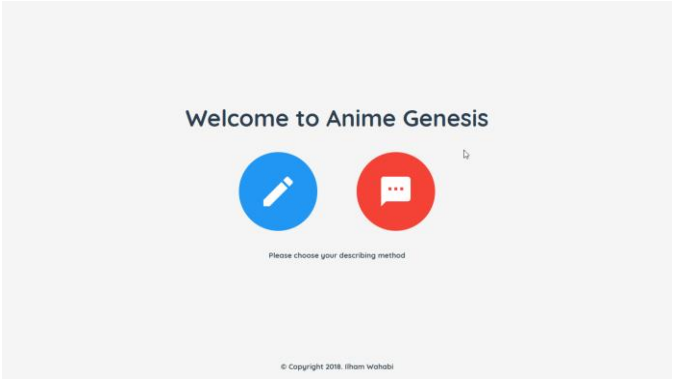
Pendahuluan

Anime merupakan salah satu produk kebudayaan jepang yang sedang popular saat ini yang menyuguhkan tontonan cukup menarik dengan berbagai macam kategori.

Saat makalah ini dituliskan sangat banyak kategori anime yang tersedia serta terlalu banyak cara untuk menyaringnya baik itu berdasarkan genre, judul, karakter yang ada, rating, staff, bahkan pengisi suara. Sungguh tidak praktis jika mengeksplorasi anime secara manual pada saat sekarang ini, apalagi kedepannya daftar anime tentunya akan semakin banyak maka kedepannya hasil pencarian akan semakin membengkak.

Maka penulis mencoba membuat aplikasi pencarian anime berdasarkan deskripsi yang diinput pengguna. Kenapa tidak menggunakan fitur lain semisal dropdown dan sejenis? Disni penulis ingin mencoba melakukan pencarian dengan algoritma pencocokan string yang sudah dipelajari, penulis juga berpikir dengan banyaknya filter yang dapat dipasang untuk melakukan pencarian suatu anime itu akan mengakibatkan tampilan web akan cukup berantakan karena tentunya kita mestinya banyak menyediakan pilihan dropdown. Dan zaman sekarang pengguna lebih suka hal yang tampilannya lebih simpel dan polos.

Inputan nantinya akan diterima baik melalui kolom teks maupun dari perintah suara. Dan akhirnya akan menampilkan anime yang sesuai ke pengguna. Versi ini bisa dibilang merupakan versi yang sangat simple dibandingkan versi yang bisa dibilang menggunakan machine learning, kecerdasan buatan, dan semacamnya. Yang bisa mendeteksi kalimat yang ada hingga ke detail terkecilnya. Sehingga besar kemungkinan aplikasi ini akan mengandung banyak sekali *bug* mengingat kita hanya mengandalkan teknik pencocokan string konvensional.



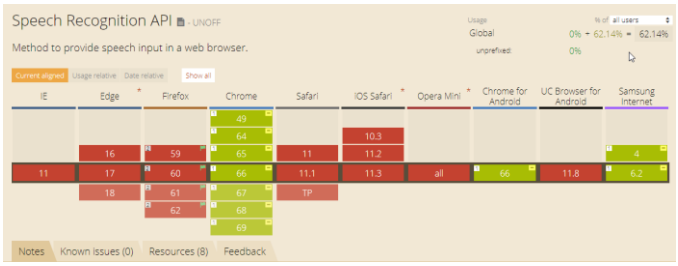
Tampilan depan situs web anime genesis

I. PENGGUNAAN

Pertama, buka laman web pada link berikut ini https://anime-genesis.firebaseapp.com/ kemudian pilih metode penginputan yang diinginkan. Disini tersedia 2 pilihan yakni melalui inputan teks maupun inputan suara. Kemudian tekan tombol pencarian untuk memulai pemrosesan. Pemrosesan dilakukan sepenuhnya disisi klien (pengguna) sehingga tidak melibatkan pemrosesan server.

II. DASAR TEORI

Salah satu fitur yang baru saja dimunculkan di perambah terbaru khususnya perambah Google Chrome (versi 25+) adalah Web Speech API yang fitur ini membuka kesempatan lebih banyak untuk pengembang mewujudkan banyak kemungkinan untuk aksesibilitas dan mekanisme pengontrolan situs web. Saat pertama kali menerima masukan suara dari mikrofon perangkat, suara itu dideteksi terlebih dahulu apa grammar yang digunakan barulah mengambalikan keluaran berupa string. Sehingga kita bisa mengembangkan aplikasi layaknya Cortana (Windows 10), Dictation (Max OS X), Siri (iOS), Android Speech, dan sejenis.



Dukungan perambah pada fitur SpeechRecognition

Sayangnya API ini sendiri saat penulis sedang menulis makalah ini masih harus diakses secara daring atau *online* karena suara tersebut harus dikirim dan diproses terlebih dahulu di mesin pengenalan suara milik Google. Semoga kedepannya fitur ini tersedia secara luring alias *offline* sehingga aplikasi yang menggunakan API bisa lebih cepat dalam menanggapi masukan. Karena fitur ini sangant membantu bagi penyandang disabilitas, contohnya mengisi formulir cukup dengan suara saja serta dapat menjalankan aplikasi tersebut sambil berkendara cukup dengan memberikan perintah suara saja sehingga mata bisa tetap fokus ke jalannan.

Sebenarnya untuk melakukan hal ini telah ada berbagai teknologi yang sudah dikembangkan sebelumnya seperti VoiceXML dan Java Applet. Sayangnya keduanya memiliki banyak masalah. Contohnya, yang pertama mereka Cuma bisa mendukung perambah Opera (yang mana notabene penggunanya kalah jauh dibandingkan pengguna Chrome, Firefox, dan Safari saat penulis sedang menulis makalah ini) perlu mengirim data ke server terlebih dahulu (karena kode yang bisa dieksekusi di sisi klient hanya Javascript) dan dieksekusi berdasarkan aksi yang telah ditentukan oleh pengguna. Hal ini sangat membutuhkan banyak baris kode dan waktu yang lebih. Untungnya dengan disediakannya fitur ini pada akhir 2012 pengembang bisa sedikit lega.

Dimana API ini sendiri terdiri dari 2 bagian yaitu:

1. SpeechSyntesis (Teks ke suara)

   Yang mana object yang dibentuk dari interface ini mempunyai metode sebagai berikut:

   a. onstart, menangani saat browser mulai mendengar suara
   b. onresult, menangani saat browser mengembalikan hasil
   c. onerror. menangani saat browser mengirim pesan kesalahan
   d. onend, menangani saat layanan dihentikan

   Serta mempunyai property sebagai berikut:

   a. continuous, menentukan tipe pengenalan suara apakah one-shot atau berlanjut
   b. lang, menentukan bahasa apa yang digunakan
   c. interimResults: menentukan apa akan menerima hasil sementara atau hasil keseluruhan

2. SpeechRecognition (Suara ke teks)

Pada makalah kali ini, penulis memakai SpeechRecognition Interface sehingga kita bisa menerjemahkan inputan suara ke teks untuk selanjutnya diolah dan dilakukan string matching.

Sedangkan, disisi lain Ppncocokan *string (pattern matching)* merupakan pencarian *string* di dalam teks. Persoalan pencarian *string* memiliki teks (*text*), yaitu *string* yang memiliki panjang n karakter, dan pola (*pattern*) yang merupakan *string* dengan panjang m karakter (m < n) yang akan dicari pada teks. *Pattern* matching akan mencari posisi pertama yang sesuai dengan pola di dalam teks.

Sepotong bagian dari teks dapat berupa sufiks atau prefiks. Pada *string* S yang memiliki panjang m, prefiks dari S adalah bagian dari *string* yang berada di antara posisi ke-0 hingga k dan sufiks dari S merupakan bagian dari *string* yang berada di antara posisi k, di mana k adalah indeks yang berada di antara 0 hingga m-1.

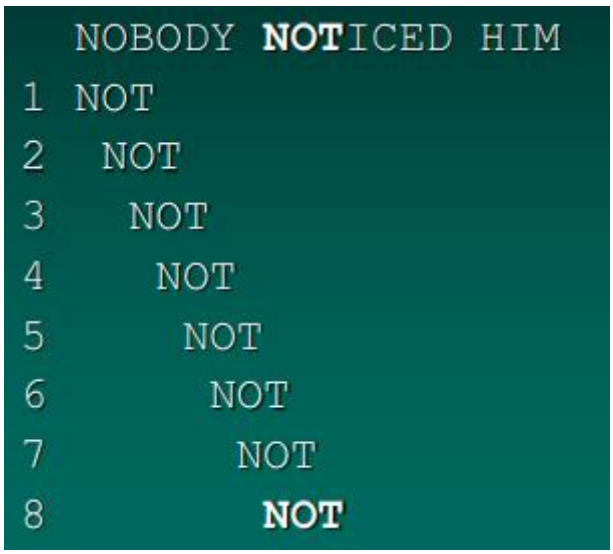
1. Algoritma Bruteforce

   Pada pencocokan *string*, algoritma *bruteforce* melakukan pencarian *string* dengan melakukan perbandingan pada setiap karakter yang dijumpai hingga menemukan bagian dari teks yang sesuai dengan pola. Bila teks berada dalam tabel karakter T[0…n] dan pola berada dalam tabel karakter

P[0…m], maka proses pencocokan *string* dengan algoritma *bruteforce* adalah sebagai berikut.

a. Indeks P (idxP) dimulai dari 0 dan indeks T (idxT) dimulai dari 0.
b. Karakter P yang berada pada posisi idxP dibandingkan dengan karakter T yang berada pada posisi idxT.
c. Jika P[idxP] dan T[idxT] cocok, maka pencarian akan dilanjutkan dengan idxP dan idxT berikutnya hingga semua karakter pada P cocok. Sedangkan apabila P[idxP] dan T[idxT] tidak cocok dan teks T belum mencapai akhir, pada pencarian selanjutnya idxP akan dimulai dari 0 kembali dan idxT maju ke idxT berikutnya.

Pada algoritma *bruteforce*, kasus terbaik yang dapat terjadi adalah ketika karakter pertama pada pola tidak pernah sama dengan karakter teks T yang dicocokkan. Jumlah perbandingan yang dilakukan maksimal n kali, sehingga kompleksitas kasus terbaiknya adalah O(n). Sedangkan, kasus terburuknya adalah ketika indeks prefix terpanjang dari pola cocok dengan setiap prefix teks yang panjangnya lebih dari atau sama dengan panjang pola, tetapi baru menemukan kecocokan pola pada akhir teks. Jumlah perbandingan yang dilakukan adalah m(n-m+1) kali, sehingga kompleksitas waktu terburuknya adalah O(mn).



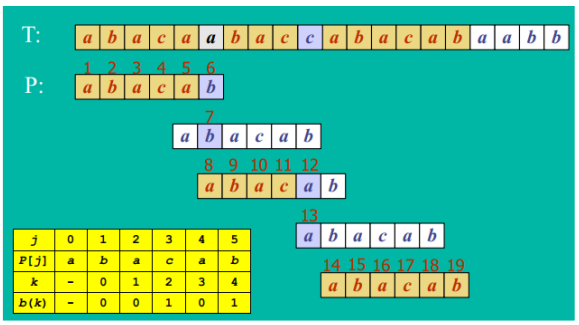
Contoh langkah penggunaan algoritma Bruteforce

2. Algoritma Knuth-Morris-Pratt

Pada algoritma KMP, pencocokan *string* juga dilakukan dengan urutan dari kiri ke kanan seperti algoritma *bruteforce*. Perbedaannya terletak dari pergeseran pola yang dilakukan ketika melakukan pencocokan. Ketika ketidakcocokan pola dan teks muncul, sistem akan menggeser pola sejauh mungkin agar tidak melakukan perbandingan berlebihan. Hal ini dilakukan dengan memanfaatkan proses prakomputasi pola untuk menemukan kecocokan prefiks pola dengan pola itu sendiri.

Prakomputasi pola pada algoritma KMP memanfaatkan fungsi pinggiran (*border function*), yaitu panjang maksimum dari prefiks pola yang juga merupakan sufiks dari pola yang memiliki panjang j.

Kompleksitas waktu algoritma KMP adalah O(m+n), karena untuk menghitung fungsi pinggiran dibutuhkan waktu sebesar O(m) dan pencarian *string* sebesar O(n)



Contoh langkah penggunaan algoritma KMP

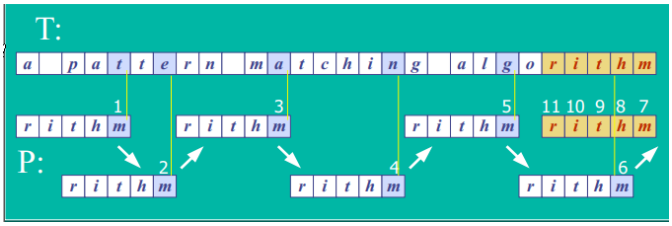
3. Algoritma Boyer-Moore

Pencocokan *string* dengan algoritma Boyer-Moore menggunakan dua teknik, yaitu *lookingglass* untuk mencari pola P pada teks T dengan mengeceknya dari akhir ke awal, dan teknik *character-jump* yang dilakukan ketika terjadi ketidakcocokan pada T[i] == P[j]. Berikut ini adalah 3 kasus ketidakcocokan yang dapat terjadi.

1) Apabila T[i] berisi elemen x dan pola P mengandung elemen x di mana posisinya berada sebelum P[j], maka pola P akan bergeser ke kanan untuk menyejajarkan posisi kemunculan terakhir x pada P dan T[i], lalu kembali memulai perbandingan dari kanan.

2) Apabila T[i] berisi elemen x dan pola P mengandung elemen x di mana posisinya berada setelah P[j], maka pola P akan bergeser satu karakter agar sejajar dengan T[i+1].

3) Pada kasus yang tidak tercakup dalam kasus 1 dan 2, pola P bergeser untuk menyejajarkan P[0] dengan T[i+1]. Algoritma Boyer-Moore melakukan prakomputasi pola P dan variasi dari setiap karakter yang ada di dalam teks T untuk membentuk fungsi kemunculan terakhir (*last occurrence function*).



Contoh langkah penggunaan algoritma Boyer-Moore

4. Regular Expression

Pada teknik regular expression kita diminta mendefinisikan sebuah pattern terlebih dahulu untuk kemudian dilakukan pencocokan dengan suatu string, sangat berguna untuk mencari kata yang 'tidak 100% sama' dengan yang diinginkan.

Contoh sintaks regular expression adalah sebagai berikut:

?, menandakan 0 atau 1 karakter
*, menandakan 0 atau lebih karakter
+, menandakan kemunculan karakter yang diikuti bisa lebih dari 1
{x}, maka karakter yang diikuti haruslah tepat sebanyak x karakter

Contoh penggunaan regular expression dapat dilihat pada contoh dibawah ini.

`[hc]?at` cocok dengan "at", "hat", and "cat".

`[hc]*at` cocok dengan "at", "hat", "cat", "hhat", "chat", "hcat", "cchchat", and so on.

`[hc]+at` cocok dengan "hat", "cat", "hhat", "chat", "hcat", "cchchat", and so on, but not "at".

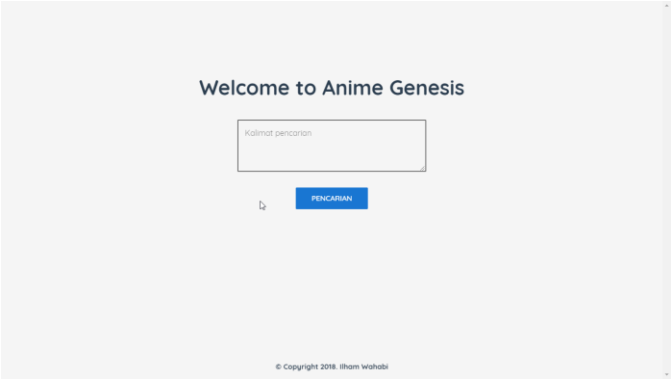
`cat|dog` cocok dengan "cat" or "dog".

Contoh situs web yang menyediakan tempat bermain regex

III. STUDI KASUS

A. *Memberi masukan*

Saat pengguna memasukkan inputan (baik melalui teks maupun suara) maka masukan tersebut akan disimpan didalam variabel untuk diproses selanjutnya. Adapun masukan suara akan dikonversi kedalam bentuk teks terlebih dahulu. Masukan suara sekarang dapat dengan mudah diterima melalui API bawaan browser modern seperti yang telah penulis jabarkan sebelumnya diatas. Adapun disini penulis menggunakan library bantu untuk menerima masukan suaranya.



Formulir memberi masukan kalimat yang akan diolah

var input = "anime mecha dengan rating lebih dari 8"

B. Cari berdasarkan kategori

Sebelumnya, kita dapatkan terlebih dahulu kategori yang ada melalui API dan disimpan didalam suatu variable larik. Kemudian dilakukan string matching pada masukan yang telah dimasukkan pengguna apakah terdapat dalam larik kategori. Jika iya maka masukkan variabel hasil.

```
var categories = [‘action’, ‘romance’, ‘mecha’, …]
var input = “anime mecha dengan rating lebih dari 8”

// String matching dengan Booyer-Moore
anime mecha dengan rating lebih dari 8
action
…
anime mecha dengan rating lebih dari 8
                                    action // tidak valid
…..
anime mecha dengan rating lebih dari 8
romance
…
anime mecha dengan rating lebih dari 8
                                  romance // tidak valid
…..

anime mecha dengan rating lebih dari 8
mecha

anime mecha dengan rating lebih dari 8
  mecha

anime mecha dengan rating lebih dari 8
        mecha // Jackpot
```

C. Cari berdasarkan hal lain

Bisa jadi pengguna tidak hanya memasukkan masukan berupa kategori yang diminta tapi juga bisa judul, rating, serta tokoh yang ada pada suatu anime. Maka disini kita mengidentifikasi masukan tersebut juga dan mencarinya berdasarkan variabel hasil yang telah didapatkan tadi.

```
var input = “anime mecha dengan rating lebih dari 8”

// String matching dengan Regular Expression

// Didapatkan kata rating

// Setelahnya didapatkan kata lebih dari 8

// Mencari di variabel hasil tadi yang animenya
//mempunyai rating > 8

// Memperbaharui variabel hasil tadi dengan hasil
//yang lebih sesuai
// Dengan menyaring hasil sebelumnya
```

D. Menampilkan ke pengguna

Setelah mendapatkan variabel hasil yang telah disimpan dalam larik maka akan ditampilkan kesisi pengguna. Ditampilkan dalam bentuk ‘kartu’. Disini penulis menggunakan framework VueJS untuk memudahkan pengembangan aplikasi di sisi kliennya.

IV. KESIMPULAN

Pencarian pada API berdasarkan inputan pengguna dapat dilakukan dengan berbagai algoritma string matching yang ada yaitu Bruteforce, Knuth-Morris-Pratt, Boyer-Moore, dan Regular Expression. Awalnya, inputan yang didapat dari teks atau suara dicari kategorinya terlebih dahulu. Untuk menerima inputan suara penulis menggunakan library bantu yang sudah mengimplementasikan fungsionalitas untuk mengakses Web Speech API yang sudah disediakan oleh perambah modern. Perintah suara dapat diinput baik melalui mikrofon ataupun langsung ke desktop. Bisa di desktop ataupun di perangkat saku.

Disini penulis menggunakan Boyer-Moore. Kemudian dicari pencarian lainnya berdasarkan informasi tambahan seperti judul, rating, dan sebagainya disini penulis menggunakan Regular Expression. Barulah hasil tersebut ditampilkan ke sisi pengguna. Dengan semuanya diolah disisi pengguna maka akses akan lebih cepat dibandingkan harus dikirim kesisi server terlebih dahulu. Apalagi jika menggunakan teknologi web terbaru yang sudah lebih cepat dan lebih handal.

REFERENSI

[1] https://anilist.gitbooks.io/anilist-apiv2-docs/, accessed on 13 May 2018. (*references*)

[2] https://en.wikipedia.org/wiki/Regular_expression, accessed on 13 May 2018.

[3] https://www.sitepoint.com/introducing-web-speech-api/, accessed on 14 May 2018.

[4] https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API/Using_the_Web_Speech_API, accessed on 14 May 2018.

[5] Dr. Andrew Davison, Pattern Matching presentation. Updated by: Dr. Rinaldi Munir.

[6] Munir, Rinaldi. 2004. Diktat IF2211 Strategi Algoritmik: Algoritma Pencarian String (*String Matching*)

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, 14 May 2018

Ilham Wahabi
13516141