Implementasi *String Matching* dengan *Regular Expressions* pada Google Form

Putri Nurhaliza - 13520066
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13520066@std.stei.itb.ac.id

***Abstrak*— Dengan perkembangan teknologi yang semakin canggih, formulir *online* menjadi sebuah kakas yang sangat membantu untuk memenuhi kebutuhan informasi. Informasi yang dibutuhkan seringkali bervariasi. Mulai dari nama, nomor telepon, email, deskripsi, hingga informasi yang bersifat pribadi seperti password. Beberapa layanan menyediakan halaman tersendiri untuk mengumpulkan informasi penggunanya. Di sisi lain, dapat digunakan kakas yang sudah ada seperti Google Form agar pembuat formulir tidak perlu repot-repot membuat layanan khusus. Untuk mencegah pengguna memasukkan sembarang informasi pada formulir yang telah disediakan, perlu dilakukan validasi terlebih dahulu sebelum data tersebut ditambhakan ke database. Pada makalah ini, dipaparkan implementasi *String matching* dengan *Regular Expressions* pada google form, serta implementasinya bersama algoritma Boyer-moore pada pengembangan web secara umum.**

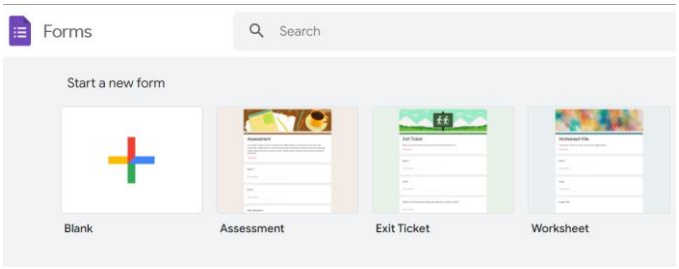
Kata kunci— Regular Expressions, string matching, form

I. PENDAHULUAN

Formulir merupakan sebuah media pengumpulan informasi yang dimanfaatkan oleh berbagai pihak untuk berbagai kepentingan. Meluasnya penggunaan formulir tidak lain adalah karena formulir sangat bermanfaat untuk meningkatkan efisiensi pelaksanaan operasional kerja. Formulir memungkinkan keseragaman dalam pembuatan, penggunaan, penyimpanan, dan pemeliharannnya. Dengan pekembangan teknologi informasi, penggunaan perangkat komputer memberikan pengarh signifikan terhadap penggunaan dan standarisasi formulir. Perancangan formulir menjadi lebih mudah untuk disesuaikan dengan kebutuhan. Modifikasi formulir juga dapat dilakukan dengan mudah dan cepat.

Form sangat sering kita jumpai dalam satu aplikasi sistem informasi berbasis web. Pada aplikasi berbasis web, form digunakan untuk menerima masukan dari pengguna. Selanjutnya, masukan dari pengguna tersebut diolah menggunakan bahasa pemrograman web, baik dengan *script* di *server-side* (misalkan PHP, JSP) ataupun *script* di *client-side* (javascript). Form dapat digunakan untuk berbagai keperluan seperti keperluan login, transaksi penjualan, mengumpulkan informasi atau meminta umpan balik dari pengguna, menawarkan barang/jasa secara on-line dan sebagainya.

Google Form merupakan salah satu layanan terbaik untuk membuat survei atau kuesioner online saat ini sehingga sudah tidak asing bagi banyak orang. Seperti yang kita ketahui, Google Form merupakan layanan yang tersedia secara gratis dari Google untuk para penggunanya. Banyak perusahaan maupun individu yang menggunakan google form untuk membangun hubungan dengan pelanggan atau pengguna. Google form adalah layanan dari Google yang memungkinkan kita untuk membuat survey dan tanya jawab dengan fitur formulir *online* yang bisa dikustomisasi sesuai dengan kebutuhan. Google Form memudahkan kita untuk membuat sekaligus mengkoleksi atau mengumpulkan data pengguna. Google Form menyediakan pilihan template formulir seperti Blank, Assessment, Worksheet, Event Feedback, Order Form, JobApplication, Time Off Request, dll. Terdapat beberapa format pertanyaan, seperti pilihan ganda, jawaban singkat, jawaban panjang, dan kotak centang yang dapat disesuaikan dengan kebutuhan.



Gambar 1. Halaman Utama Google Form

II. TEORI DASAR

A. *Bruteforce*

Pencocokan String dilakukan pada dua string, dengan salah satu string sebagai *text* dengan panjang n karakter, dan satu string lainnya adalah pattern dengan panjang m karakter. Pada implementasinya, digunakan asumsi m < n. Pada algoritma brute force, dilakukan perbandingan satu per satu karakter pada pattern dengan teks, dari kiri ke kanan. Terdapat dua kemungkinan. Pertama, kika hasil perbandingan karakter sama, maka cek karakter selanjutnya pada pattern dengan indeks selanjutnya pada teks. Kedua, jika ditemukan ketidakcocokan pada perbandingan terakhir maka geser pattern satu karakter ke kanan teks. Lalu, akan dilakukan

perbandingan lagi mulai dari karakter pertama pada pattern. Perbandingan akan terus dilakuakan secara iteratif hingga ditemukan kecocokan. Jika perbandingan telah mencapai indeks terakhir teks dan indeks terakhir pattern lalu ditemukan ketidakcocokan karakter maka dapat dinyatakan pattern tersebut tidak ditemukan pada teks.

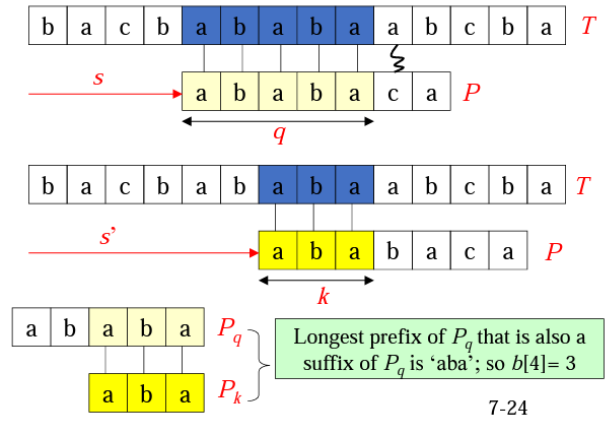
```
Teks: NOBODY NOTICED HIM
Pattern: NOT

  NOBODY NOTICED HIM
1 NOT
2  NOT
3   NOT
4    NOT
5     NOT
6      NOT
7       NOT
8        NOT
```

Gambar 2. String Matching dengan Algoritma Bruteforce

Performa algoritma ini akan semakin baik dan cepat jika semakin banyak variasi karakter pada suatu teks. Kasus terburuk terjadi ketika ketidakcocokan karakter selalu terjadi pada ujung pattern. Misalnya, ketika teks adalah "xxxxxxxxxxy" dan pattern-nya adalah "xxy", kita perlu melakukan pergeseran pattern mencapai ujung teks dan setiap pergeseran perlu dilakukan hingga karakter terakhir pada pattern. Jumlah perbandingan yang dilakukan adalah m(n-m+1) sehingga kompleksitasnya adalah O(mn). Kasus terbaik terjadi ketika ketidakcocokan selalu terjadi pada indeks pertama pada pattern, misalnya ketika teks adalah "qwertyuiopaaa" dan pattern-nya adalah "aaa". Pada kasus terbaik jumlah perbandingan maksimal adalah n kali sehingga kompleksitasnya O(n). Pada pencarian teks normal, pada umumnya ditemuo kasus rata-rata yang memiliki kompleksitas O(m+n).

B. Knuth-Morris-Pratt (KMP)

Algoritma KMP merupakan perbaikan dari algoritma bruteforce dimanan pergeseran pattern dilakukan sedemikian rupa sehingga jumlah perbandingan yang dilakukan menjadi lebih sedikit. Jika terjadi ketidakcocokan antara teks dan pattern, misalkan $T[i] \neq P[j]$, maka pergeseran pattern dapat dilakukan pada prefix dari $P[0\ldots j-1]$ yang merupakan suffix dari $P[1\ldots j-1]$. Pergeseran ini akan mengurangi operasi perbandingan. Algoritma KMP dibantu dengan sebuah fungsi pinggiran atau *border function*, dinotasikan dengan b(k), dimana k adalah nomor indeks pada pattern. b(k) didefenisikan sebagai ukuran prefix dari $P[0\ldots k]$ terbesar yang merupakan suffix $P[1\ldots k]$ dengan $k = j-1$ dan $b(k)$ sebagai nilai j yang baru. Algoritma KMP mengembalikan indeks dimana pattern dimulai pada text jika ditemukan. Jika tidak, akan mengembalikan nilai -1.



Gambar 3. String Matching dengan Algoritma Knuth-Morris-Pratt

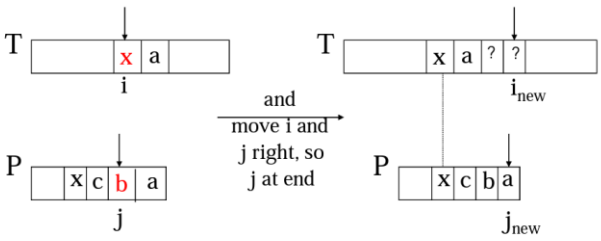
Misal pada sebuah pattern "ababa" maka prefiksnya adalah "a", "ab", "aba", "ababa", sedangkan suffiksnya adalah "a", "ba", "aba", "baba". Jika akan dicari nilai pinggiran untuk k = 4, pada suffiks dan prefiks ditemukan kesamaan pada "aba". Maka fungsi b(4) akan mengembalikan nilai panjang "aba", yaitu 3.

Kompleksitas waktu pada fungsi pinggiran adalah O(m). Sementara itu, komplesitas yang dibutuhkan untuk pencarian string adalah O(n) sehingga kompleksitas total dari algoritma KMP adalah O(m+n). Algoritma KMP sangat cepat relatif terhadap algoritma bruteforce. Namun, jika fungsi pinggiran mengembalikan nilai yang kecil atau bahkan 0, maka kemangkusannya akan berkurang.

C. Boyer-Moore

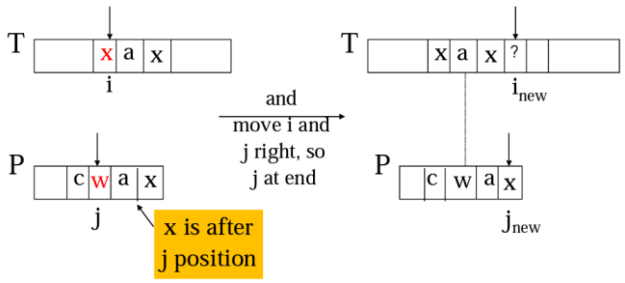
Algoritma Boyer-Moore menggunakan konsep yang agak berbeda dibanding algoritma sebelumnya. Pencocokan string dilakukan dari kanan ke kiri. Algoritma ini menggunakan dua metode, yaitu *looking-glass technique* dan *character-jump technique*. P ditemukan pada T dengan bergerak mundur melalui P, mulai dari ujung akhir. Lalu, character-jump dilakukan saat terjadi ketidakcocokan saat $T[i] = x$ dan karakter di pattern $P[j] \neq T[i]$. Terdapat 3 kasus yang mungkin, yaitu sebagai berikut.

- Jika P memiliki x, maka geser P ke kanan hingga kemunculan terakhir x di P sejajar dengan $T[i]$.



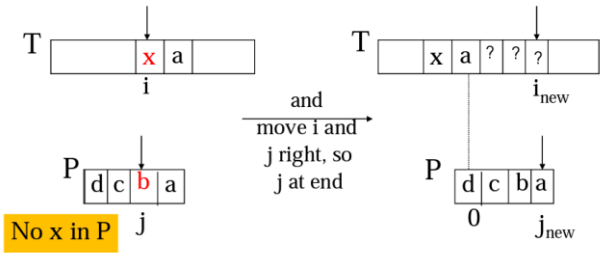
Gambar 4. Kasus 1 Algoritma Boyer-Moore

- Jika P memiliki *x*, namun tidak memungkinkan melakukan pergeseran di atas, maka geser *P* satu karakter ke kanan (*T*[*i*+1]).



Gambar 5. Kasus 2 Algoritma Boyer-Moore

- Jika tidak memenuhi kedua kasus di atas, maka geser *P* sehingga *P*[0] sejajar dengan *T*[*i*+1].



Gambar 6. Kasus 3 Algoritma Boyer-Moore

kompleksitas algoritma Boyer-Moore adalah O(nm + A). Kasus terburuk terjadi ketika teks sangat mirip dengan parttern, dan ketidakcocokan ditemukan di karakter pertama. Contohnya ketika teksnya "bbbbbbbbbb" dan patternnya "abb". Performa algoritma Boyer-Moore akan semakin baik performansinya jika alfabetnya semakin bervariasi.

D. Regular Expressions

Regular Expressions (regex) adalah notasi standar yang mendeskripsikan suatu pola (*pattern*) berupa urutan karakter atau string. Regex dapat mengenali string dengan karakteristik dan pola tertentu, misal email. nomor identitas, tanggal, dan lain lain. Notasi regex dapat dikombinasikan untuk pencocokan string dengan pola yang cukup rumit.

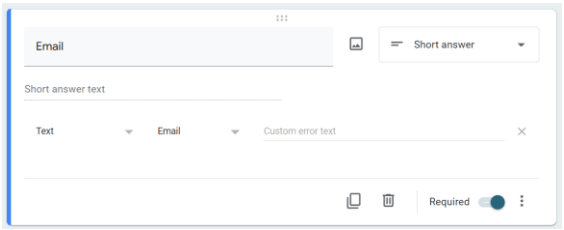
Tabel 1. Notasi *Regular Expressions*

Wildcard (‘,’)	Cocok dengan satu karakter apapun. regex : /n.p/ match : nlp, n3p, dll.
Optionality ‘?’	Menandakan bahwa regex yang diberikan sebelum simbol tersebut bersifat opsional. regex : /colou?r/ match :colour dan color.
Repeatability ‘+’ dan ‘*’	Simbol ‘+’ menandakan bahwa regex yang diberikan sebelum simbol tersebut dapat diulang. regex : /coo+l/ match : cool, coool, cooool, dst Simbol ‘*’ menandakan bahwa regex yang diberikan sebelum simbol tersebut bersifat opsional atau dapat berulang.
	regex : /coo*l/ match : col, cool, coool, dst.
Choice ‘[]’	Pola kata yang cocok terbatas pada regex yang diberikan di dalam simbol tersebut. regex : /n[a,l,o]p/ match : nap, nlp, dan nop
Range ‘-‘	Menunjukkan suatu range untuk karakter. regex : /n[a-z]p/ match : nap, nbp, ncp, … , nzp
Complementation ‘^’	Digunakan sebagai negasi. regex : /^[aiueo]/ match : string dengan karakter konsonan saja
Common special symbol ‘^’ dan’$’	Menandai awalan dan akhiran dari sebuah baris.
Other special character	\b : *word boundary* \d : *decimal digit* ([0-9]) \D : *non-digit character* ([^0-9]) \s : *whitespace character* ([\t \n \r \f \v]) \S : *non-whitespace character* ([^ \t \n \r \f \v]) \w : *alphanumeric character* ([a-zA-Z0-9_]) \W: any *non-alphanumeric character* ([^a-zA-Z0-9_])

III. IMPLEMENTASI DAN PEMBAHASAN

Sistem validasi formulir *online* pada sebuah website sebaiknya terjadi secara langsung untuk memproses setiap input dari pengguna. Validasi perlu dilakukan untuk memastikan konsistensi data yang masuk. Untuk mengoptimalkannya, akan dimanfaatkan pesan kesalahan jika data yang diinput tidak sesuai dengan yang diharapkan. Hal ini dapat mempermudah pengguna untuk mengetahui kesalahannya berdasarkan kriteria input yang diharapkan tanpa perlu mencoba-coba atau memuat ulang halaman formulir. Google Forms menyediakan fitur untuk validasi masukan pengguna. Sebagai pembuat form, kita dapat memanfaatkannya sebaik mungkin dengan menentukan aturan-aturan tertentu. Validasi yang sering dijumpai saat melakukan input data biasanya adalah validasi bahwa input tidak boleh kosong.

Salah satu jenis inputan yang paling sering dibutuhkan adalah email sehingga penting untuk divalidasi. Email valid berisi beberapa karakter sebelum tanda “@”, serta domain email di akhir email. Google form sendiri sudah menyediakan pilihan untuk email sehingga pembuat form tidak perlu mengspesifikkan regex yang digunakan. Fitur tersebut dapat diaksses dengan memilih pilihan jawaban singkat, lalu menambahkan validasi respons.



Gambar 7. Fitur validasi format masukan pengguna pada Google Form

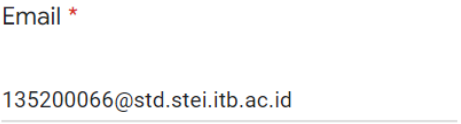
Sementara itu, untuk implementasi pada aplikasi web secara umum, *Regular Expressions* yang digunakan pada validasi email adalah

/^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0- 9-.]+$/

Regular Expressions di atas secara rinci sebagai berikut.

- ^ menandakan awal string
- [a-zA-Z0-9_.+-] semua huruf termasuk kapital dan angka serta karakter "_", ".", "+", "-".
- +@ menyatakan minimal terdapat satu karakter atau lebih sebelum sebuah karakter "@".
- [a-zA-Z0-9-] semua huruf termasuk kapita dan angka serta karakter "-".
- +\. menyatakan minimal terdapat satu karakter atau lebih sebelum karakter ".".
- $ menandakan akhir dari sebuah string

(a)



(b)

Gambar 8. Validasi Email (a) gagal, (b) berhasil

Pada bahasa pemograman javascript, suatu string dapat diuji apakah sesuai dengan *Regular Expression* menggunakan fungsi test. Pertama-tama, kita harus mendefenisikan pola yang diinginkan terlebih dahulu.

```
let pattern = /(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9])(?=.*?[!@#\><*~]).{8,}/;
```

Selanjutnya, pola tersebut diuji dengan fungsi *test* untuk memeriksa validitas email yang diinput.

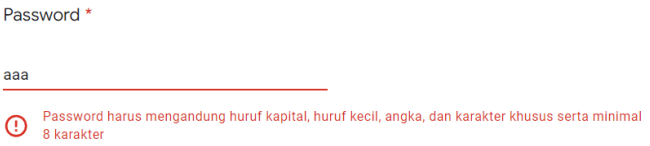
```
const email = prompt("Email: ");
if (pattern.test(email)) {
    console.log("Email valid");
} else {
    console.log("Email tidak valid");
}
```

Kombinasi password yang kompleks diperlukan untuk meningkatkan keamanan. Salah satu kombinasi yang sering digunakan adalah kombinasi huruf kapital, huruf kecil, angka, dan karakter khusus, serta minimal 8 karakter. *Regular Expressions* yang dapat digunakan adalah sebagai berikut.

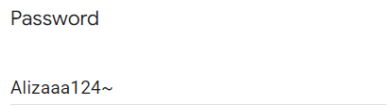
/(?=.*?[A-Z])(?=.*?[a-z])
(?=.*?[0-9])(?=.*?[!@#\><*~]).{8,}/

Regular Expressions di atas secara rinci sebagai berikut.

- (?=.*?[A-Z]), mengandung minimal 1 huruf kapital
- (?=.*?[a-z]), mengandung minimal 1 huruf kecil
- (?=.*?[0-9]), mengandung minimal 1 angka
- (?=.*?[!@#\><*~]), mengandung minimal 1 karakter khusus
- {8,}, panjang password minimal adalah 8 karakter



(a)



(b)

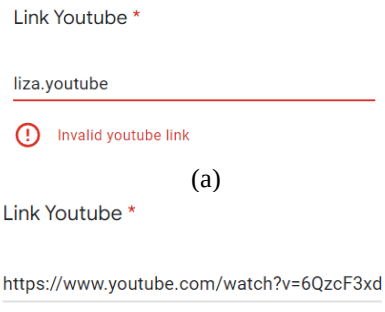
Gambar 9. Validasi Password (a) gagal, (b) berhasil

Selain itu, seringkali dibutuhkan URL dari pengguna, misal link menuju akun media sosial maupun menuju postingan tertentu. Tujuan dari validasi URL tersebut adalah memerikasa inputan yang dimasukkan oleh user, sudah sesuai atau belum dengan standar penulisan URL. Untuk memastikan link yang diinput pengguna memenuhi aturan format link yang valid, dapat digunakan *Regular Expressions* sebagai berikut.

/((http|https)://)(www.)?[a-zA-Z0-9@:%._\\+~#?&//=]{2,256}\\.[a-z]{2,6}\\b([-a-zA-Z0-9@:%._\\+~#?&//=]*)/

Format URL juga dapat dibuat lebih spesifik. Misal, untuk link youtube dapat digunakan *Regular Expressions* sebagai berikut.

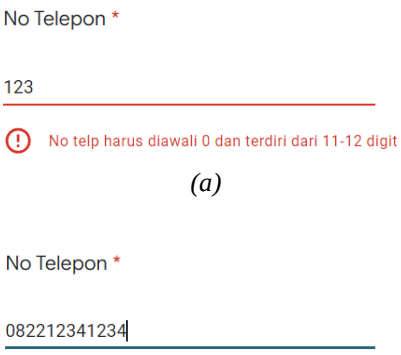
/https?\:\/\/(www\.)?youtu(\.)?be(\.com)?\/.*(\?v=|\/v\/)?[a-zA-Z0-9_\-]+/



Gambar 10. Validasi URL (a) gagal, (b) berhasil

Nomor telepon juga merupakan salah satu jenis inputan yang sering dibutuhkan sebagai informasi kontak pengguna. Tidak hanya itu, beberapa layanan juga menggunakan nomor telepeon sebagai ID autentikasi. Untuk menghindari format nomor telepon yang mengandung simbol-simbol ekstra dari seperti strip atau spasi, maka dapat digunakan *Regular Expressions* agar selanjutnya tidak perlu dilakukan normalisasi. Misal, jika pengguna diharapkan untuk memasukkan 11-12 digit dimulai dengan angka nol atau +62, dapat digunakan *Regular Expressions* berikut.

/^[\+62,0]([0-9]{10,11})$/



Gambar 11. Validasi nomor telepon (a) gagal, (b) berhasil

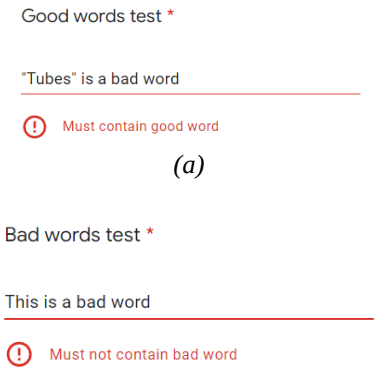
Pembuat form dapat memastikan inputan pengguna mengandung suatu kata tertentu. Setidaknya terdapat dua cara ynag dapat dilakukan. Pertama, menggunakan *Regular Expressions*. *Regular Expressions* yang dapat digunakan misal

/good word/

dengan "good word" dapat digantikan dengan kata yang diinginkan. Selain itu, Pembuat form juga dapat memastikan inputan pengguna tidak mengandung suatu kata tertentu. *Regular Expressions* yang dapat digunakan misal

/^((?!bad word).)*$/

dengan "bad word" dapat digantikan dengan kata yang ingin dikecualikan.



Gambar 12. Pembatasan contains or not contains pada teks

Sayangnya, *Regular Expressions* juga memiliki keterbatasan dalam mengenali suatu pola. Regex yang kompleks masih memberikan peluang terjadinya kesalahan, dimana pola yang seharusnya tidak cocok dianggap cocok atau sebaliknya. Permasalahan *contains or not contains* juga dapat diimplementasikan dengan algoritma string matching bruteforce, Knuth-Morrris-Pratt, maupun Boyer-Moore. Jika diinginkan harus terdapat kata Y dan diinginkan tidak ada kata N, Boyer-More diiimplementasikan dengan menjadikan kata Y dan N sebagai pattern, dan inputan keseluruhan sebagai text.

```
// Algoritma utama Boyer-Moore
const boyerMoore = (text, pattern) => {
   const last = buildLast(pattern);
   let n = text.length;
   let m = pattern.length;
   let i = m-1;

   if (i > n-1) {
      return -1;
   }

   let j = m-1;
   do {
      if (text.charAt(i) === pattern.charAt(j)){
         if (j === 0) {
            return i;
         } else {
            i--;
            j--;
         }
      } else {
         const lo = last[text.charAt(i)];
         i = i + m - Math.min(j, 1 + lo);
         j = m - 1;
      }
   } while (i <= n-1);

   return -1;
};
```

Program di atas adalah implementasi algoritma Boyer-Moore pada bahasa pemograman javascript. Algoritma Boyer-Moore dibantu dengan sebuah fungsi untuk menemukan indeks i sehingga P[i] == X dimana X adalah sebuah karakter pada kata.

```
const buildLast = (pattern) => {
    const last = new Array(128);
    for (let i = 0; i < 128; i++) {
        last[i] = -1;
    }

    for (let i = 0; i < pattern.length; i++) {
        last[pattern.charAt(i)] = i;
    }

    return last;
};
```

Algoritma Boyer-Moore di atas akan mengembalikan -1 jika kata tidak ditemukan pada teks, dan sebaliknya akan mengembalikan indeks karakter pertama kata jika kata ditemukan pada teks. Fungsi ini dapat dimanfaatkan pada validasi inputan. Misal, jika pencocokan string antara teks dan kata N mengembalikan nilai selain -1, maka akan ditampilkan pesan kesalahan sebab kata N seharusnya tidak terdapat pada teks input. Berikut adalah contoh program sederhana pemanfaaan algoritma Boyer-moore pada kasus di atas.

```
if (boyerMoore(text, pattern) == -1) {
    console.log("Succes, your input is valid");
} else {
    console.log("Your input is invalid, can not
contains bad word");
}
```

```
text: "this is a bad word"
must not contains: "bad word"
Your input is invalid, can not contains bad word
```

Gambar 13. Contoh Output Pesan Kesalahan dengan Algoritma Boyer-moore

IV. KESIMPULAN

Validasi format inputan pada formulir *online* termasuk Google Form dapat diaplikasikan dengan *Regular Expressions*. Regex digunakan untuk mengecek inputan pengguna secara langsung untuk menjaga kebersihan *database*. Dengan memanfaatkan teknik diatas, pengguna akan terbantu dengan mengetahui error yang terjadi seketika mungkin. Sayangnya, *Regular Expressions* juga memiliki keterbatasan sehingga belum mampu sepenuhnya mampu mengecek validitas suatu masukan, khususnya untuk format yang kompleks. Notasi *Regular Expressions* yang terlalu panjang akan sulit dimengerti. Oleh sebab itu, algoritma Bruteforce, Boyer-Moore, ataupun KMP dapat digunakan sebagai alternatif sebab menggunakan logika yang lebih sederhana. *Regular Expressions* yang diimplementasikan pada pembuatan Google Form di atas dapat diimplementasikan pada pengembangan web lain dengan berbagai bahasa pemograman. Perlu diperhatikan bahwa beberapa bahasa pemograman memiliki sintaks yang agak berbeda untuk *Regular Expressions* sehingga diperlukan sedikit penyesuaian.

VIDEO LINK YOUTUBE

Pemaparan isi makalah ini dapat dilihat pada pranala berikut.

https://youtu.be/RbW7arUUN8k

UCAPAN TERIMAKASIH

Pertama-tama, penulis memanjatkan puji dan syukur atas kehadirat Tuhan Yang Maha Esa yang telah memberikan berkat dan rahmat yang tak terhingga, sehingga akhirnya penulis dapat menyelesaikan makalah ini dengan baik. Penulis mengucapkan terima kasih kepada semua pihak yang telah mendukung penulis dalam proses penulisan makalah ini. Khususnya kepada seluruh dosen Kelas Matematika Diskrit IF2210, yaitu Bapak Rinaldi Munir, Ibu Nur Ulfa Maulidevi, dan Ibu Masayu Leylia Khodra beserta seluruh asisten yang telah mengajarkan dan membagi ilmunya yang diperlukan dalam penulisan makalah ini.

DAFTAR PUSTAKA

[1] Munir, Rinaldi (2021). Pencocokan String String/Pattern Matching. Diakses pada 18 Mei 2022 (https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf)

[2] Khodra, Masayu L (2019). String Matching dengan *Regular Expressions*. Diakses pada 18 Mei 2022 (https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf)

[3] Bird, S., & Klein, E. (2006). *Regular Expressions* for natural language processing. University of Pennsylvania.

[4] Bulat, R (2020). *Regular Expressions* in JavaScript: An Introduction. Diakses pada 18 Mei 2022 (https://rossbulat.medium.com/regular-expressions-in-javascript-an-introduction-94a40dce46a2)

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, 22 Mei 2022

Putri Nurhaliza