

# Penerapan String Matching dan Regular Expression dalam Pembuatan ChatGPT Sederhana

Tugas Besar III IF2211 Strategi Algoritma Semester II Tahun 2022/2023



## **Bab I**

### **Deskripsi Tugas**

#### **Latar belakang:**

Dalam dunia teknologi, chatbot telah menjadi hal yang umum digunakan dalam berbagai aplikasi dan platform seperti situs web, aplikasi mobile, dan media sosial. Chatbot memungkinkan pengguna untuk berinteraksi dengan program yang memiliki kemampuan untuk memproses dan merespons percakapan secara otomatis. Salah satu contoh chatbot yang sedang booming saat ini adalah ChatGPT. Pembangunan chatbot dapat dilakukan dengan menggunakan berbagai pendekatan dari bidang Question Answering (QA). Pendekatan QA yang paling sederhana adalah menyimpan sejumlah pasangan pertanyaan dan jawaban, menentukan pertanyaan yang paling mirip dengan pertanyaan yang diberikan pengguna, dan memberikan jawabannya kepada pengguna. Untuk mencocokkan input pengguna dengan pertanyaan yang disimpan pada database, kalian bisa menggunakan string matching. String matching adalah teknik untuk mencocokkan suatu string atau pola dengan string lainnya, dengan tujuan untuk menentukan apakah kedua string tersebut cocok atau tidak. Teknik ini biasanya digunakan dalam chatbot untuk mengenali kata-kata atau frasa tertentu yang dapat dipahami oleh program dan digunakan sebagai input untuk menentukan respon yang sesuai. Sementara itu, regular expression adalah kumpulan aturan atau pola yang digunakan untuk pencocokan string dengan format yang spesifik. Teknik ini sering digunakan dalam chatbot untuk mengenali dan memproses input pengguna yang memiliki format tertentu, seperti nomor telepon, alamat email, atau kode pos.

#### **Deskripsi tugas:**

Dalam tugas besar ini, anda diminta untuk membangun sebuah aplikasi ChatGPT sederhana dengan mengaplikasikan pendekatan QA yang paling sederhana tersebut. Pencarian pertanyaan yang paling mirip dengan pertanyaan yang diberikan pengguna dilakukan dengan algoritma pencocokan string

Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM). Regex digunakan untuk menentukan format dari pertanyaan (akan dijelaskan lebih lanjut pada bagian fitur aplikasi). Jika tidak ada satupun pertanyaan pada database yang exact match dengan pertanyaan pengguna melalui algoritma KMP ataupun BM, maka gunakan pertanyaan termirip dengan kesamaan setidaknya 90%. Apabila tidak ada pertanyaan yang kemiripannya di atas 90%, maka chatbot akan memberikan maksimum 3 pilihan pertanyaan yang paling mirip untuk dipilih oleh pengguna. Perhitungan tingkat kemiripan dibebaskan kepada anda asalkan dijelaskan di laporan, namun disarankan menggunakan salah satu dari algoritma Hamming Distance, Levenshtein Distance, ataupun Longest Common Subsequence.

## Bab II

### Landasan Teori

- **KMP**

Algoritma Knuth-Morris-Pratt (KMP) adalah algoritma pencocokan string yang digunakan untuk mencari keberadaan sebuah pola (pattern) dalam sebuah teks (text) dengan kompleksitas waktu  $O(m + n)$ , dimana m adalah panjang pola dan n adalah panjang teks.

KMP bekerja dengan cara membangun sebuah tabel prefiks-sufiks terpanjang (Longest Prefix Suffix - LPS) dari pola yang akan dicocokkan terlebih dahulu, yang menyimpan nilai prefiks terpanjang dari setiap sub-pola dari awal hingga posisi yang sedang ditinjau dan juga nilai sufiks terpanjang dari setiap sub-pola dari akhir hingga posisi yang sedang ditinjau. Nilai-nilai dalam tabel LPS ini kemudian digunakan untuk menentukan perpindahan posisi pencocokan pola ketika terjadi ketidakcocokan pada posisi tertentu dalam teks.

Proses pembuatan tabel LPS dilakukan dengan mengiterasi pola dari awal hingga akhir, dan setiap kali terjadi ketidakcocokan, tabel LPS diperbarui dengan mengambil nilai LPS sebelumnya yang sesuai dan memeriksa kembali pola dari nilai tersebut. Jika nilai pada posisi tersebut cocok, maka nilai pada tabel LPS diupdate, jika tidak maka proses diulang hingga ditemukan nilai LPS yang cocok. Proses ini berlanjut hingga akhir dari pola.

Setelah tabel LPS selesai dibangun, proses pencocokan dimulai dari awal teks dan pola. Jika karakter pada posisi tertentu pada pola cocok dengan karakter pada posisi yang sama pada teks, maka proses pencocokan dilanjutkan ke karakter berikutnya. Namun, jika terjadi ketidakcocokan, maka posisi pencocokan pola dipindahkan ke posisi pada tabel LPS yang sesuai dengan karakter pada posisi tersebut, dan proses pencocokan dilanjutkan kembali dari posisi tersebut.

- **BM**

Algoritma Boyer-Moore (BM) adalah algoritma pencocokan string yang digunakan untuk mencari keberadaan sebuah pola (pattern) dalam sebuah teks (text) dengan kompleksitas waktu  $O(m + n)$ , dimana m adalah panjang pola dan n adalah panjang teks.

Algoritma BM bekerja dengan cara menggeser pola untuk mencocokkannya dengan teks dari kanan ke kiri, sehingga ketika terjadi ketidakcocokan pada posisi tertentu, kita dapat memanfaatkan informasi tentang pola tersebut dan teks yang sedang dicocokkan untuk mempercepat proses pencarian. Informasi ini diperoleh dari dua tabel yaitu tabel karakter kanan (rightmost occurrence) dan tabel pergeseran baik-buruk (bad character shift).

Tabel karakter kanan menyimpan informasi tentang kemunculan terakhir sebuah karakter dalam pola, sehingga ketika terjadi ketidakcocokan pada posisi tertentu dalam teks, kita dapat memanfaatkan informasi ini untuk memindahkan posisi pencocokan pola sejauh mungkin ke kanan, sehingga dapat mengurangi jumlah karakter yang perlu dicocokkan lagi.

Sedangkan tabel pergeseran baik-buruk (bad character shift) menyimpan informasi tentang posisi terakhir kemunculan sebuah karakter pada pola, sehingga ketika terjadi ketidakcocokan pada posisi tertentu dalam teks, kita dapat memanfaatkan informasi ini untuk mempercepat proses pergeseran posisi pencocokan pola. Jika karakter pada posisi tertentu dalam teks tidak cocok dengan karakter pada posisi yang sama dalam pola, kita dapat menggeser pola sejauh mungkin ke kiri hingga karakter tersebut menjadi sejajar dengan karakter pada posisi yang cocok dalam teks.

Dengan kombinasi dari tabel karakter kanan dan tabel pergeseran baik-buruk, algoritma BM memungkinkan pencarian sebuah pola dalam sebuah teks dengan kompleksitas waktu yang efisien, karena jumlah karakter yang perlu dicocokkan ulang dapat diurangi secara signifikan pada setiap langkah pencarian.

- **Regex**

Regex (Regular Expression) adalah sebuah string khusus yang digunakan untuk mencocokkan atau mencari pola tertentu dalam sebuah teks. Regex dapat digunakan pada banyak bahasa pemrograman dan tool seperti Perl, Python, Java, sed, awk, dan lain sebagainya.

Regex terdiri dari karakter-karakter khusus yang memiliki makna tertentu, seperti huruf, angka, spasi, tanda baca, dan simbol. Beberapa karakter khusus yang sering digunakan dalam regex antara lain:

^ : digunakan untuk mencocokkan awal baris

\$ : digunakan untuk mencocokkan akhir baris

. : digunakan untuk mencocokkan satu karakter apa saja kecuali karakter newline (\n)

: digunakan untuk mencocokkan nol atau lebih karakter sebelumnya

: digunakan untuk mencocokkan satu atau lebih karakter sebelumnya

? : digunakan untuk mencocokkan nol atau satu karakter sebelumnya

[] : digunakan untuk mencocokkan satu karakter dari kumpulan karakter yang diberikan

() : digunakan untuk grup dan sub-pola dalam pencocokan

| : digunakan untuk mencocokkan salah satu pola yang diberikan

Regex dapat digunakan untuk berbagai macam keperluan, seperti pencarian teks pada dokumen, validasi input pada form, penggantian string tertentu dengan string lain, dan sebagainya.

Penggunaan regex juga dapat dioptimalkan dengan teknik-teknik seperti lookahead, lookbehind, dan quantifiers yang memungkinkan kita untuk memanfaatkan pola yang lebih kompleks dan spesifik.

## Bab III

### Implementasi Program

- **KMP**

Algoritma KMP diimplementasikan sebagai berikut



```
1 package StringMatching
2
3 func BMMatch(text, pattern string) int {
4     last := buildLast(pattern)
5     n, m := len(text), len(pattern)
6     if m > n || m == 0{
7         return -1 // no match if pattern is longer than text
8     }
9     i, j := m-1, m-1
10    for i < n {
11        if pattern[j] == text[i] {
12            if j == 0 {
13                return i // match
14            }
15            i--
16            j--
17        } else {
18            lo := last[text[i]] // last occ
19            i += m - min(j, 1+lo)
20            j = m - 1
21        }
22    }
23    return -1 // no match
24 }
25
26 func buildLast(pattern string) [128]int {
27     if len(pattern) == 0 {
28         return [128]int{}
29     }
30     var last [128]int // ASCII char set
31     for i := range last {
32         last[i] = -1 // initialize array
33     }
34     for i := 0; i < len(pattern); i++ {
35         last[pattern[i]] = i
36     }
37     return last
38 }
39
40 func min(a, b int) int {
41     if a < b {
42         return a
43     }
44     return b
45 }
46
```

- **BM**

Algoritma BM diimplementasikan sebagai berikut

```
1 package StringMatching
2
3 func BMMatch(text, pattern string) int {
4     last := buildLast(pattern)
5     n, m := len(text), len(pattern)
6     if m > n || m == 0{
7         return -1 // no match if pattern is longer than text
8     }
9     i, j := m-1, m-1
10    for i < n {
11        if pattern[j] == text[i] {
12            if j == 0 {
13                return i // match
14            }
15            i--
16            j--
17        } else {
18            lo := last[text[i]] // Last occ
19            i += m - min(j, 1+lo)
20            j = m - 1
21        }
22    }
23    return -1 // no match
24 }
25
26 func buildLast(pattern string) [128]int {
27     if len(pattern) == 0 {
28         return [128]int{}
29     }
30     var last [128]int // ASCII char set
31     for i := range last {
32         last[i] = -1 // initialize array
33     }
34     for i := 0; i < len(pattern); i++ {
35         last[pattern[i]] = i
36     }
37     return last
38 }
39
40 func min(a, b int) int {
41     if a < b {
42         return a
43     }
44     return b
45 }
46
```

- **Regex**

Regex (Regular Expression) adalah sebuah string khusus yang digunakan untuk mencocokkan atau mencari pola tertentu dalam sebuah teks. Regex dapat digunakan pada banyak bahasa pemrograman dan tool seperti Perl, Python, Java, sed, awk, dan lain sebagainya.

Regex terdiri dari karakter-karakter khusus yang memiliki makna tertentu, seperti huruf, angka, spasi, tanda baca, dan simbol. Beberapa karakter khusus yang sering digunakan dalam regex antara lain:

^ : digunakan untuk mencocokkan awal baris  
\$ : digunakan untuk mencocokkan akhir baris  
. : digunakan untuk mencocokkan satu karakter apa saja kecuali karakter newline (\n)  
: digunakan untuk mencocokkan nol atau lebih karakter sebelumnya  
: digunakan untuk mencocokkan satu atau lebih karakter sebelumnya  
? : digunakan untuk mencocokkan nol atau satu karakter sebelumnya  
[] : digunakan untuk mencocokkan satu karakter dari kumpulan karakter yang diberikan  
| : digunakan untuk mencocokkan salah satu pola yang diberikan

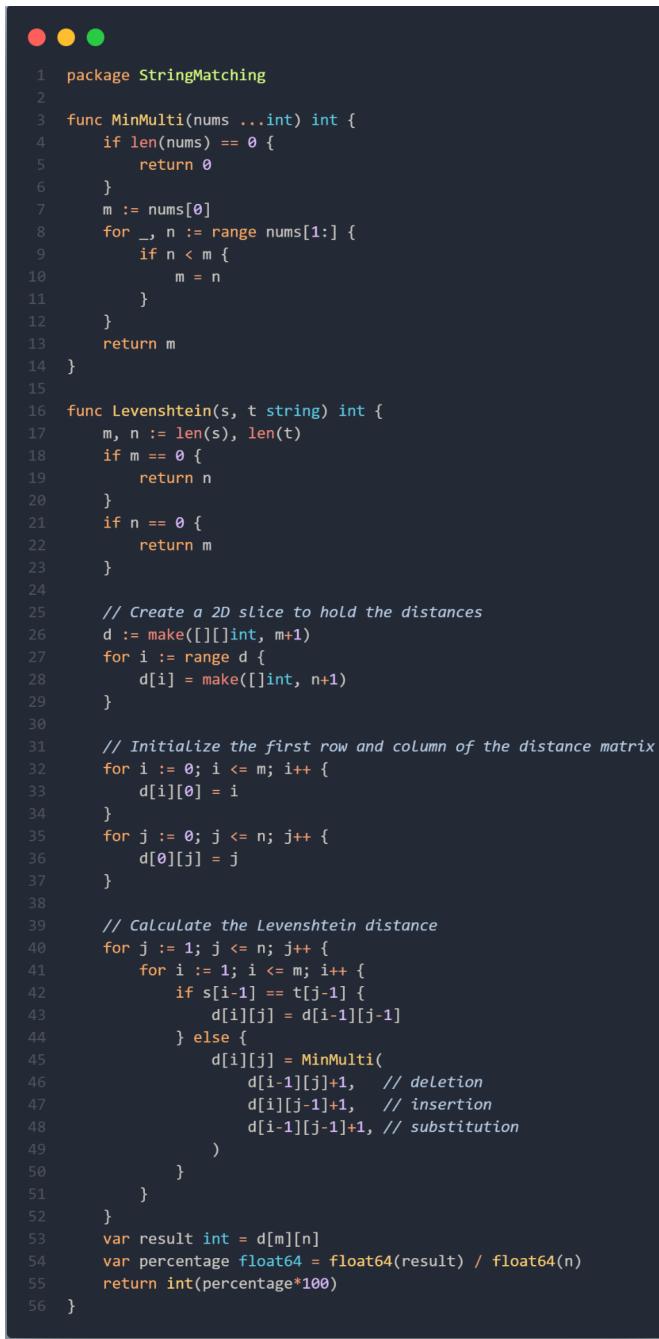
Regex dapat digunakan untuk berbagai macam keperluan, seperti pencarian teks pada dokumen, validasi input pada form, penggantian string tertentu dengan string lain, dan sebagainya.

Penggunaan regex juga dapat dioptimalkan dengan teknik-teknik seperti lookahead, lookbehind, dan quantifiers yang memungkinkan kita untuk memanfaatkan pola yang lebih kompleks dan spesifik.

- Levenshtein Distance

Adapun Levenshtein Distance untuk mencari kemiripan dari kedua string diimplementasikan

sebagai berikut



```
● ● ●
1 package StringMatching
2
3 func MinMulti(nums ...int) int {
4     if len(nums) == 0 {
5         return 0
6     }
7     m := nums[0]
8     for _, n := range nums[1:] {
9         if n < m {
10            m = n
11        }
12    }
13    return m
14 }
15
16 func Levenshtein(s, t string) int {
17     m, n := len(s), len(t)
18     if m == 0 {
19         return n
20     }
21     if n == 0 {
22         return m
23     }
24
25     // Create a 2D slice to hold the distances
26     d := make([][]int, m+1)
27     for i := range d {
28         d[i] = make([]int, n+1)
29     }
30
31     // Initialize the first row and column of the distance matrix
32     for i := 0; i <= m; i++ {
33         d[i][0] = i
34     }
35     for j := 0; j <= n; j++ {
36         d[0][j] = j
37     }
38
39     // Calculate the Levenshtein distance
40     for j := 1; j <= n; j++ {
41         for i := 1; i <= m; i++ {
42             if s[i-1] == t[j-1] {
43                 d[i][j] = d[i-1][j-1]
44             } else {
45                 d[i][j] = MinMulti(
46                     d[i-1][j]+1,      // deletion
47                     d[i][j-1]+1,    // insertion
48                     d[i-1][j-1]+1, // substitution
49                 )
50             }
51         }
52     }
53     var result int = d[m][n]
54     var percentage float64 = float64(result) / float64(n)
55     return int(percentage*100)
56 }
```

## **Lampiran**

Repository: [https://github.com/christodharma/Tubes3\\_13521005](https://github.com/christodharma/Tubes3_13521005)