Algoritma Minimal Edit Distance serta Regular Expression untuk Membersihkan Teks dari Kata Kasar

Muhammad Risqi Firdaus - 13520043
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13520043@std.stei.itb.ac.id

***Abstract*—Semakin berkembangnya teknologi, maka sosialisasi pun beralih melalui media digital. Teks merupakan salah satu media sosialisasi yang banyak digunakan. Pada sistem digital, teks disimpan dalam memory.**
Tidak dapat dipungkiri, perkembangan dunia menyebabkan kata-kata pada teks sangat variatif dan muncul penggunaan kata kasar ketika bersosialisasi di melalui media digital.
Oleh karena itu, dibuatlah sebuah program filtrasi teks menggunakan dua algoritma, regular expression serta minimum edit distance atau levenshtein.

Keywords—regular expression, distance, kata kasar, string.

Pendahuluan (*HEADING 1*)

Bahasa merupakan alat yang tidak dapat dipisahkan dari manusia. Bahasa merupakan cara manusia satu dengan yang lain berkomunikasi. Melalui bahasa terjadinya tukar informas, tukar rasa, serta sosialisasi antarmanusia.

Salah satu cara penyampaian informasi ialah melalui komunikasi tertulis. Komunikasi ini menjadikan susan huruf yang terpampang sebagai media bertukar ifnormasi.

Metode tertulis tak jarang kurang dapat menyampaikan sentimennya, sehingga pada kondisi tertentu ditambahkanlah kata-kata untuk memberikan kesan pada tulisan tersebut. Kesan yang diberikan, tidak selalu merupakan kesan yang positif, tak jarang, kesan yang disampaikan justru berisi kata-kata yang kurang ramah, atau kata-kata kasar.

Bahasa atau perkataan yang iducapkan sesseorang tidak selalu mengandung kata yang baik. Tidak jarang, seseorang menggunakan kata kasar atau kata yang kurang baik untuk berkomunikasi satu sama lain.

Kata kasar bukan lah kata yang ramah bagi setiap orang. Bagi sebagian orang, kata kasar dapat menjadi ungkapan yang melukai hati, atau tak jarang mengintimidasi.

Di era digital seperti saat ini, pengungkapan kata atau tulisan tak lagi hanya melalui bentuk cetak, tetapi juga melalui hasil ketikan atau pesan digital.

Pesan digital sejatinya merupakan kumpulan memory yang tersusun menjadi kata-kata, untuk kemudian dibaca sebagai bahasa manusia.

Dengan digitalisasi, pertukaran informasi tulisan menjadi lebih mudah, lebih bebas, dan lebih cepat. Digitalisasi sejatinya telah mengubah bentuk komunikasi tertulis dengan cepat. Melalui hal tersebut, huruf-huruf disusun, agar dibaca orang lain, agar tersampaikan berbagai pesan.

Dibandingkan dengan tulisan cetak yang sangat susah untuk di ubah, susunan kata atau huruf pada tulisan digital lebih mudah diubah daripada tulisan cetak. Kita dapat mengubah susunan digital semudah mengubah isi memori yang menyimpan tulisan tersebut.

Tak jarang kita menemui kata yang tak pantas pada tulisan digital. Sudah menjadi keharusan, untuk menghindari kerusakan akibat kata-kata yang tak pantas pada tulisan digital. Sebagai insan akademis yang berbudi, kata tak pantas harus selalu dihindari dalam komunikasi.

Oleh karena itu, perlu sebuah program atau algoritma untuk mengeliminasi potensi kata kasar pada sebuah tulisan digital. Dengan demikian, diharapkan penggunaan kata kasar dapat diminimalisasi.

I. LANDASAN TEORI

A. Pattern Matching

Pattern Matching adalah proses pencocokan string atau pola. Pattern matching akan mencari kecocokan bentuk atau kesamaan bentuk dari sebuah pola di dalam sebuah teks. Pada zaman modern seperti saat ini, pattern matching banyak sekali diaplikasikan di dunia nyata, seperti validasi masukan pada form, algoritma pencarian, pencocokan DNA dan rantai protein serta masih banyak lagi.

Berikut contoh proses pattern matching, dengan teks T dan patter P.

T : string = “AKU SUKA MAKAN NASI GOYENG”
P: string = “ASI”

Algoritma yang didesain akan mencoba mencari mengecek keberadaaan dari pola “ASI” di dalam teks T. Jika terdapat pola P dalam teks T maka algoritma akan mengembalikan true, sebaliknya algoritma akan mengembalikan false.

Dengan pattern matching, juga dapat dicek, di manakah posisi pola pada teks. Pada contoh di atas, pola terdapat pada indeks ke-16 hingga 17 pada teks.

B. Brute Force Algorithm

Brute Force merupakan algoritma yang paling kotor. Algoritma ini menyelesaikan masalah dengan mencoba semua kemungkinan yang ada. Dengan begitu, kompleksitas dari algoritma ini selalu sangat tinggi. Meskipun sangat tidak efektif, algoritma ini hampir selalu bisa menyelesaikan semua masalah.

Misalnya pada pattern matching, algoritma ini akan menjajal semua kombinasi karakter yang dapat dibuat antara string yang dicocokkan dan pola yang ada. Kondisi terburuknya, terjadi ketika program mencocokkan pola dengan bagian paling akhir dari kombinasi string dan pola.

C. Decrease and Conqueror

Decrease and Conquer adalah metode desain algoritma dengan mereduksi persoalan menjadi beberapa sub-persoalan yang lebih kecil. Berbeda dengan Divide and Conquer yang memproses semua sub-persoalan dan menggabungkan tiap solusi, algoritma Decrease and Conquer hanya mereduksi sebuah sub persoalan kecil dan memproses salah satu sub-persoalan saja.

Metode ini terdiri dari dua tahapan:

1. Decrease merupakan tahap mereduksi persoalan menjadi persoalan yang lebih kecil
2. Conquer merupakan tahap pemrosesan satu sub-persoalan secara rekursif.

Yang membedakan algoritma ini dengan algoritma divide and conquer adalah tidak adanya tahapan combine pada akhir algoritma decrease and conquer.

Algoritma decrease and conquer dapat dibedakan menjadi tiga jenis, yakni,

1. Decrease by a constant merupakan algoritma decrease and conquer yang setiap iterasinya direduksi dengan sebuah nilai konstan, seperti 1, 2, dan sebagainya.
2. Decrease by a constant factor merupakan algoritma decrease and conquer yang tiap instans persoalan direduksi dengan faktor konstanta yang sama, berbeda dengan decrease by a constant, decrease by a constant factor mungkin terlihat lebih seperti dibagi, dibandingkan dengan decrease by a constant yang proses pengurangannya dikurangi secara literal,.
3. Decrease by a variable size merupakan algoritma decrease and conquer yang untuk setiap iterasinya dikurangi dengan sebuah variabel tertentu.

Komplesitasnya algoritma ini ditentukan sebagaimana algorima rekursi lainya, berdasarkan T(n) yang terdefinisi dari algoritma yang disusun.

D. Algoritma *Minimal Edit Distance (Heading 2)*

Minimum edit distance atau levenshtein distance merupakan algoritma yang kerap digunakan untuk menghitung nilai perbedaan dari dua buah string. Algoritma ini akan menghitung banyak operasi yang diperlukan pada salah satu string agar kedua string yang dibandingkan menjadi identik.

Berbeda dengan Hamming distance yang memerlukan panjang string yang sama, levenshtein dapat diaplikasikan pada string dengan panjang berbeda. Algoritma ini akan mengenumerasi seluruh operasi yang dapat dilakukan pada kedua string.

Sebuah operasi didefinisikan sebagai menyisipkan sebuah karakter, menghapus sebuah karakter atau menukar sebuah karakter. Algoritma ini akan mencari operasi paling sedikit yang diperlukan agar kedua string identik.

		k	i	t	t	e	n
	0	1	2	3	4	5	6
s	1	1	2	3	4	5	6
i	2	2	1	2	3	4	5
t	3	3	2	1	2	3	4
t	4	4	3	2	1	2	3
i	5	5	4	3	2	2	3
n	6	6	5	4	3	3	2
g	7	7	6	5	4	4	3

Gambar 1.3 Matrix perhitungan Minimum Edit Distnce

Algoritma yang paling mudah untuk menerapkan minimum edit distance adalah menggunakan metode Rekursi dengan Decrease and Conqueror. Misalnya, terdapat dua buah string s1 dan s2. Kemudian, s1 ingin diubah menjadi s2. Maka dapat diterapkan tiga pilihan menukar, menghapus atau menambahkan karakter pada s1.

Penerapan yang paling umum digunakan untuk algoritma ini ialah dengan metode rekursi. fungsi rekursinya pun didefinisikan sebagai berikut.

$$f(N,M) = \begin{cases} N & M = 0 \\ f(N-1, M-1) & N = 0 \\ 1 + \text{minimum} \begin{pmatrix} f(N-1,M-1) \\ f(N-1,M) \\ f(N,M-1) \end{pmatrix} & \text{otherwise} \end{cases}$$

Basis dari fungsi ini terdiri dari 3 persamaan, yakni ketika panjang salah satu string sama dengan nol, atau ketika head (karakter pertama) dari string, sama. Selanjutnya rekurens dari fungsi ini adalah satu ditambah nilai minimal dari 3 kombinasi pemotongan kepala karakter kedua string.

Jika diperhatikan dengan seksama. Algoritma ini *fit* dengan bentuk dari algoritma Decrease and Conqueror by constant

dengan nilai constant 1, itulah mengapa pada awal bagian disebutkan bahwa algoritma ini termasuk brute-force. Yakni karena Kompleksitas dari algoritma ini berada pada kisaran O(NxM). Hal ini disebabkan algoritma Minimal edit distance akan menghitung semua kemungkinan dari operasi yang dapat dilakukan kepada kedua string.

Untuk mendapatkan pendekatan nilai dari similarity string yang dicek, dapat ditambahkan operasi pembagian dengan panjang string terbesar, kemudian digunakan untuk menggurangi bilangan satu dengan nilai tersebut.

$$sim(N,M) = \begin{cases} 1 - f(N,M)/M & M > N \\ 1 - f(N,M)/N & M < N \end{cases}$$

Formula 1.2 mencari nilai kemiripan dengan konstanta jarak

E. Regular Expression

Regular expression atau yang kerap disebut regex merupakan suatu notasi dan karakter yang digunakan untuk mendapatkan sebuah susunan karakter melalui pola yang dibuat. Regex merupakan salah satu metodestring matching yang paling mangkus. Untuk setiap operasinya hanya memakan waktu 1 operasi.

Regeular expression cukup efektif jika ingin digunakan untuk mencocokkan string yang tidak exact. Kelebihan dari regular expression sendiri akan menyaring string berdasarkan kecocokan terhadap pola regular expression.

Secara umum, regex berbentuk kumpuran karakter yang membentuk pola tertentu. Karakter yang dimaskud di sini bisa berupa tanda baca maupun huruf yang melambangkan notasi tertentu.

Regular expression sendiri terdiri atas tiga kumpulan aturan karakter khusus, yakni,

1. Character Classes
   Character classes merupakan penggambaran dari karakter tertentu, seperti digit, alfanumerik, space, newline bahkan null.

Pattern	Arti
.	Semua karakter kecuali newline
\d	Semua digit (Arabic numeral), ekuivalen [0-9]
\D	Semua karakter non-digit (Arabic numeral)
\w	Semua karakter alfanumerik, ekuivalen dengan [A-Za-z0-9_]
\W	Semua karakter yang bukan karakter pada kata dari alfabet Latin.
\s	Semua karakter single ‘white space’
\S	Semua karakter selain single ‘white space’
\t	Semua tab horizontal
\r	Carriage return
\n	New line
\v	Vertical tab
\0	null character

2. Quantifiers
   Quantifiers mengindikasikan jumlah karakter yang akan dicocokkan.

Pattern	Arti
.*	kejadian lebih besar sama dengan 0 kali
+	kejadian lebih besar sama dengan 1? kali
?	kejadian terjadi 0 atau 1
{n}	jumlah kejadian n kali
{n,m}	jumlah kejadian n hinngga m kali
{,n}	jumlah kejadian lebih besar sama dengan n
*?	kejadian 0 atau satu kali sesedikit mungkin
+?	kejadian lebih besar sama dengan satu kali sesedikit mungkin

3. Grouping
   Grouping mengelompokkkan karakter sesuai dengan group yang pola yang digrup. Group juga dapat digunakan untuk mengecualikan pattern yang digrupnya dengan notasi ‘^’.

Pattern	Arti
x|y	mencocokkan x atau y
[xyz]	Mencocokkan salah satu di antara x, y atau z
[x-z]	mencocokkan karakter antara karakter x hingga z.
[^xyz]	Bukan merupakan karakter x, y, atau z
[^x-z]	Bukan merupakan karakter antara x hingga z
(xyz)	mencocokkan seluruh karakter merupakan xyz atau tidak sama sekali
(?:xyz)	Mencocokkan xyz keseluruhan atau tidak sama sekali tapi tidak mengingatnya

II. Pembahasan

Proses pengecekan string pada program ini menggunakan dua metode secara terpisah. Gunanya agar dapat dibandingkan bagaimana efektivitas dari kedua metode, yakni regular expression dan string distance dengan algoritma minimal edit distance.

Program dibuat dengan masukan berupa sebuah file txt dengan kata-kata kasar. Program akan meload file dengan nama file sesuai dengan masukan dari pengguna. Setelah melakukan perubahan, program akan meng-*output*-kan ke-layar kata-kata yang sudah dibersihkan.

Dengan algoritma regex, perlu di-*set*, untuk setiap kosa kata yang mungkin, bagaimana formula regular expression yang cocok dengan kata tersebut. Misal, untuk kata “anjing”, dapat diformulasikan regular expression yang cocok adalah “[4aA][n]*[jy][ie31]*n*g”.

Pada program ini, daftar regular expression kata kasar disimpan dalam file .txt, agar lebih mudah ketika ingin melakukan perubahan, maupun penambahan regular expression kata kasar yang baru.

```
f = open("badRegex.txt",'r')
listOfRe = f.readlines()
f.close()

for i in range(len(listOfRe)):
    listOfRe[i] = listOfRe[i].replace("\n","").split(":")


for j in listOfRe:
    text = re.sub(j[0],j[1],text)

return text
```

Program akan melakukan iterasi pada setiap regular expression kata kasar. Dalam iterasi tersebut, program akan mencocokkan untuk setiap kata pada string dengan regular expression kata kasar. Pada kata yang sesuai, kata tersebut akan diganti dengan persamaan

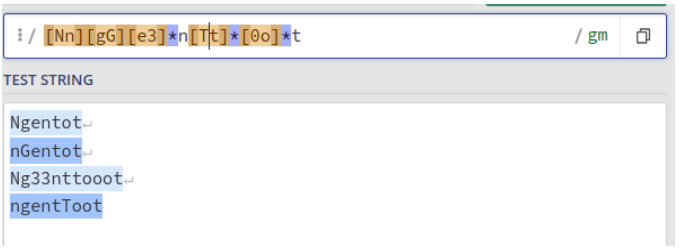
(kata[0]+”*” * (len(kata)-2)+kata[-1])

Program ini berjalan dengan kompleksitas O(n) dengan n merupakan banyaknya regular expression kata kasar yang disimpan.

Kelebihan dari program yang dibuat dengan regular expression ini adalah dapat menebak kata dengan bentuk yang memiliki perbedaan signifikan dengan bentuk awal. Kata seperti “Anyyiiiiiiingg” dapat ditebak dengan regular expression. Kata ini bisa jadi luput jika menggunakan metode minimal edit distance. Hal tersebut terjadi karena, jika dicek jaraknya jauh dari kata kasar dasar.

Kelebihan dari regular expression yang fleksibel ini pun tidak gratis, ada harga yang harus dibayar, yakni kita perlu men-*set* untuk tiap kata kasar, pola regular expression yang dapat mencakup berbagai bentuk kata kasar, seperti perubahan huruf dengan angka, serta pemanjangan atau penyingkatan kata.

Untuk meng-handle persoalan tersebut, dapat digunakan karakter “*” serta grouping dengan “[]” yang mengindikasikan kelipatan 0 atau lebih dari karakter yang digrup. Dengan demikian, perpanjangan string dengan memperbanyak karakter, atau mengubah huruf ke angka dapat di-*handle*.



Berikut adalah contoh regex untuk mem-filter kata kasar pada teks.

```
[Nn]g[e3]*nt[0o]*t:n*****t
[Kk][0o]*nt[0o]*l:k****l
[Nn]g[3e]*w[3e]*:n***e
[4aA][n]*[jy][ie31]*n*g:a****g
[Bb][4a]*[nN]*[gG]*[sS5][a4]*[tT]:b*****t
```

Selain membutuhkan skill komputasi, kemampuan peka terhadap kata juga diperlukan untuk menentukan regular expression yang tepat.

Fungsi selanjutnya adalah filter kata kasar menggunakan minimal edit distance atau levenshtein distance. Fungsi minimal edit distance akan menghitung berapa jarak antara string yang dicek dengan data base string kata kasar secara brute force. Tiap kata dari string akan dihitung jaraknya. Kemudian akan dirumuskan nilai similaritas dengan rumus (ii).

Dilakukan dua percobaan dengan konstanta kedekatan 0.8 dan 0.6. Hasilnya pada kedekatan 0.6 didapatkan bahwa kata yang dipanjangkan dan sedikit dirubah masih dapat dicover oleh fungsi ini. Sedangkan, pada konstanta 0.8 kata yang dicover lebih sedikit, dan lebih mirip dengan pola kata kasar. Namun, pada konstanta ini program tidak dapat meng-cover kata yang dipanjangkan, sebagai gantinya, progam akan menghiraukan kata wajar yang mirip dengan kata kasar seperti kata "kental".

```
def getDistance(s1,s2):
    if(len(s2)==0):
        return len(s1)
    elif(len(s1)==0):
        return len(s2)
    elif(s1[0]==s2[0]):
        return getDistance(s1[1:],s2[1:])
    else:
        return 1+min(getDistance(s1[1:],s2),min(getDistance(s1,s2[1:]),getDistance(s1[1:],s2[1:])))
```

Program ini menggunakan algoritma decrease and conquer dengan decrease by constant bernilai konstanta 1. Dengan begitu, dapat dirumuskan nilai T(n) = T(n-1) + n. Dengan nilai T(n) yang demikian, maka didapat nilai O(n) pada operasi mencari jarak sebesar O(n^2).

```
for i in ts:
    for j in listOfRe:
        maxLen = (len(j) if len(j)>len(i) else len(i))
        if getDistance(i,j)/ maxLen <= 0.4 :
            text = text.replace(i,j[0]+"*"*(len(j)-2)+j[-1]
```

Program berjalan dengan mencari kecocokan antara kata kasar dan tiap kata dalam string, sehingga nilai O(n) awal dikalikan dengan nilai O(n) iterasi yakni O(lxm) dengan l banyak kata kasar yang disimpan sistem dan m adalah banyak kata pada string.

Dari program, didapatlah nilai total O(n) = O(n^2xlxm). Karena jumlah kata kasar yang disimpan sistem sangat kecil, maka dapat diabaikan, sehingga nilai O(n) = O(n^2xm).

Kelebihan dari program ini ialah tidak perlu didefinisikan secara khusus tiap kata kasar beserta kemungkinan bentuknya karena program akan secara otomatis mendeteksi berdasarkan kata kasar dasar yang disimpan sistem.

Di sisi lain, simplisitas program ini menyebabkan ketidakmampuan program untuk meng-*handle* kata dengan pola yang cukup jauh dari kata dasar, dengan kata lain, pencocokan string dengan pola snagat bergantung pada koefisien kecocokan. Semakin tinggi nilainya, maka pola yang jauh akan semakin tidak ter-cover, sedangkan semakin kecil nilainya, bentuk kata yang berbeda arti semakin mungkin ditebak program sebagai kata kasar.

Program di tes menggunakan string

"Ngentooot, apa sih anjing, ngewe yuk, bangsat, anjing lu tai kucing, kental, anying, anjir gila ini mah"

Berdasarkan hasil pengujian didapatkan string hasil sebagai berikut,

1. Regular Expression
   n*****t, apa sih a****g, n***e yuk, b*****t, a****g lu kucing, kental, a****g, anjir gila ini mah

2. Minimal Edit Distance (koefisien 0.6)
   "n*****t, apa sih a****g, n***e yuk, b*****t, a****g lu tai kucing, k****l, a****g, a****g gila ini mah"

3. Minimal Edit Distance (koefisien 0.8)
   "ngenttoooot, apa sih a****g, n***e yuk, b*****t, a****g lu kucing, kental, a****g, a****g gila ini mah"

Acknowledgment

Teks merupakan ekspresi dari manusia yang disimpan dalam buffer memori. Teks dapat dimodifikasi dengan mudah. Dengan algoritma tertentu suatu teks dapat diidentifikasi atau dikategorikan pada kondisi tertentu.

Dari hasil pengujian yang sudah dilakukan, dapat dideduksi bahwa, terdapat kelebihan dan kekurangan pada tiap program, penggunaan fungsi harus tepat sasaran, agar dapat meng-cover semua *potential bad words*.

Sebagai saran, diperlukan penelitian lebih lanjut, mengenai kombinasi penggunaan metode pada filtrasi string. Perlu adanya program yang menggunakan ketiganya pada sebuah kasus dan meng-output-kan hasil kombinasi filtrasi seluruh metode.

Dengan begitu, diharapkan program yang dihasilkan dapat lebih bersih, serta meminimalisasi nilai error pada filtrasi kata.

Video Link at Youtube

https://youtu.be/X-HhD9jnBKQ

References

[1] http://informatika.stei.itb.ac.id/~rinaldi.munir/ diakses pada 20 Mei 2022.

[2] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions/Character_Classes diakses pada 23 Mei 2022.

[3] https://slaystudy.com/levenshtein-distance/#:~:text=The%20recursive%20function%20calculates%20levenshtein,longer%20prefixes%20and%20so%20on.&text=dp(N%2C%20M)%20is,of%20length%20M%20of%20S2. diakses pada 20 Mei 2022.

[4] Navarro, Gonzalo (1 March 2001). "A guided tour to approximate string matching" (PDF). *ACM Computing Surveys*. **33** (1): 31–88.

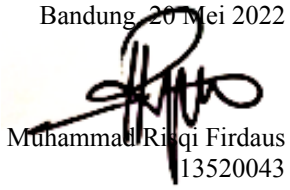
[5] Daniel Jurafsky; James H. Martin. *Speech and Language Processing*. Pearson Education International. pp. 107–111.

‘

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, 20 Mei 2022



Muhammad Rizqi Firdaus
13520043