Analisis Kemiripan Dua Bahasa dengan Approximate String Matching

Rayendra Althaf Taraka Noor - 13522107
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13522107@std.stei.itb.ac.id

***Abstract*— Penelitian ini mengkaji metode untuk menganalisis kemiripan antara dua bahasa menggunakan pendekatan approximate string matching. Approximate string matching, atau pencocokan string mendekati, merupakan teknik yang memungkinkan pencarian dan pengukuran kesamaan antara dua string meskipun terdapat perbedaan kecil seperti substitusi, penghapusan, atau penyisipan karakter. Studi ini menggunakan algoritma dalam approximate string matching yakni Levenshtein Distance membandingkan pasangan kata dan frase dari dua bahasa yang dianalisis. Hasil penelitian menunjukkan bahwa pendekatan ini cukup efektif dalam mengidentifikasi pola kemiripan dan perbedaan antar bahasa. Harapannya, penelitian ini memberikan kontribusi signifikan dalam bidang pemrosesan bahasa alami melalui penerapan approximate string matching dalam analisis kemiripan bahasa.**

Keywords—Approximate String matching; Bahasa; Levenshtein Distance; Pemrograman Dinamis

I. Pendahuluan

Sejak awal dimulainya era globalisasi, pertukaran budaya di berbagai belahan dunia banyak terjadi. Dapat diingat juga bahwa pada awal era ini sebagian dari negara-negara yang ada sekarang belum sepenuhnya terbentuk dan sebagian lainnya banyak yang mengalami revolusi setelahnya. Akibatnya terdapat banyak kemiripan budaya melekat di negara yang terbentuk sekarang. Kemiripan tersebut mungkin didapat dari negara-negara yang berdekatan ataupun negara-negara yang dulunya diduduki negara lainnya baik karena kebutuhan perang maupun penjajahan. Hingga kini, dapat diamati kemiripan-kemiripan tersebut terutama yang paling menonjol yakni kemiripan bahasa.

Bahasa merupakan salah satu bagian terbesar dari budaya yang dimiliki oleh suatu negara. Untuk melihat kemiripan antara dua bahasa, kita dapat memperhatikan tiga komponen yakni, *Pronounciation*, *Vocabulary*, dan *Grammar*. Dengan kemajuan teknologi yang ada sekarang, Kemiripan Vocabulary bisa didapat dengan memperhatikan kata untuk suatu hal yang sama dari kedua bahasa.

Dalam makalah ini, penulis ingin menunjukkan bagaimana algoritma yang telah dipelajari pada mata kuliah IF 2211 Strategi Algoritma dapat diaplikasikan untuk menganalisis kemiripan dari dua bahasa. Algoritma yang digunakan untuk bahasan ini adalah *levenshtein distance algorithm* yang merupakan salah satu algoritma untuk *approximate string matching.*

II. Dasar Teori

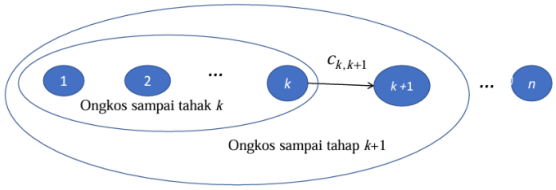
A. Approximate String Matching

Approximate string matching adalah metode mengukur seberapa mirip dua buah teks yang diberikan. Metode cukup populer untuk digunakan dalam hal-hal yang berkaitan dengan memeriksa susunan kata yang memiliki kemungkinan untuk rusak karena berbagai faktor seperti kesalahan ejaan oleh manusia dan *noise* yang muncul dalam sebuah data. Tujuan utamanya adalah untuk menemukan kecocokan antara dua *string*, bahkan jika ada sedikit.

Berbeda dengan string matching yang hanya akan memberi hasil "sama" jika kedua *string* sama persis. *Approximate string matching* memperhitungkan perbedaan dari dua buah string tersebut. Algoritma-algoritma yang digunakan untuk menyelesaikan masalah ini juga biasanya akan memberikan sebuah *distance* untuk dua buah kata yang diberikan. Umumnya nilai *distance* tersebut didapat dari penjumlahan operasi tertentu untuk mengubah sususan karakter dari dua string tersebut, Terdapat beberapa algoritma *approximate string matching* diantaranya ialah *levenshtein distance jaro-winkler distance*, dan *N-grams.*

B. Pemrograman Dinamis

Pemrograman dinamis adalah metode pemecahan masalah dengan memecahnya menjadi bagian-bagian yang dapat dikaitkan. Ruang lingkup dari pemrograman dinamis adalah *problem-problem* optimalisasi. Ide dasarnya, akan diambil sebuah nilai optimal dari semua tahapan yang mungkin dilewati untuk mencapai optimal akhir.

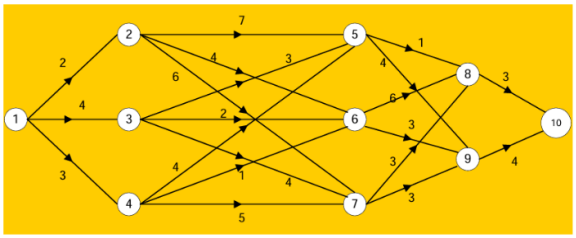


Gambar 2.2.1 gambaran per tahap pemrograman dinamis

Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Program-Dinamis-2020-Bagian1.pdf

Pemrograman dinamis dapat dibagia ke beberapa tahap untuk menyelesaikannya. Awalnya, masalah yang akan diselesaikan menggunakan algoritma ini akan dibagi menjadi tahapan-tahapan (stage) dan sejumlah status (state). Pada bagian *stage* dan *state* tertentu nantinya akan diambil sebuah keputusan/nilai untuk bagian tersebut. Kemudian nilai/keputusan tersebut akan dikaitkan ke bagian lain dengan algoritma rekursif. Nantinya, seiring dengan selesainya keberjalanan algoritma rekursif, akan ditemukan optimal akhir.

Untuk memberikan gambaran yang lebih jelas, berikut salah satu penyelesaian menggunakan pemrograman dinamis untuk lintasan terpendek dari graf multitahap.



Gambar 2.2.2 graf multitahap pemrograman dinamis

Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Program-Dinamis-2020-Bagian1.pdf

1. Pertama bagi tahap ini menjadi 5 tahapan, dari kiri, ke kanan, dan 10 status yang mewakili angka setiap node.
2. Buat hubungan rekursifnya

$$f_k(s)\begin{cases} c_{x1,s}, & k = 1 \\ \min\{f_{k-1}(x_k) + c_{xk,s}\}, & k > 1 \end{cases}$$

Huruf k mewakili nilai tahap, huruf s mewakili *state*, dan c adalah cost antar node.

3. Hitung solusi dari basis rekursi

-Tahap pertama:

	Solusi Optimum	
s	$f_1(s)$	x_1^*
2	2	1
3	4	1
4	3	1

Gambar 2.2.3 tahap pertama contoh pemrograman dinamis

Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Program-Dinamis-2020-Bagian1.pdf

-Tahap kedua:

x_2 / s	$f_2(s) = f_1(x_2) + c_{x2,s}$			Solusi Optimum	
	2	3	4	$f_2(s)$	x_2^*
5	9	7	7	7	3 atau 4
6	6	6	4	4	4
7	8	8	8	8	2, 3, 4

Gambar 2.2.4 tahap kedua contoh pemrograman dinamis

Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Program-Dinamis-2020-Bagian1.pdf

-Tahap ketiga:

x_3 / s	$f_3(s) = f_2(x_3) + c_{x3,s}$			Solusi Optimum	
	5	6	7	$f_3(s)$	x_3^*
8	8	10	11	8	5
9	11	7	11	7	6

Gambar 2.2.5 tahap ketiga contoh pemrograman dinamis

Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Program-Dinamis-2020-Bagian1.pdf

-Tahap keempat:

x_4 / s	$f_4(s) = f_3(x_4) + c_{x4,s}$		Solusi Optimum	
	8	9	$f_4(s)$	x_4^*
10	11	11	11	8 atau 9

Gambar 2.2.6 tahap keempat contoh pemrograman dinamis

Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Program-Dinamis-2020-Bagian1.pdf

C. *Levenshtein Distance*

Algoritma levenshtein distance adalah salah satu algoritma untuk *approximate string matching*. Sebagaimana algoritma *approximate string matching* pada umumnya, algoritma ini menghitung jarak dari dua buah string. Rentang nilai jarak dari hasil levenshtein distance adalah 0 sampai max(m,n) dimana m adalah panjang string pertama dan n adalah panjang string kedua.

Jarak dari dua buah string tersebut akan diukur dari total operasi penyisipan, penghapusan, dan penggantian karakter. Berikut penjelasan dari tiga buah operasi tersebut:

1. Penyisipan

   Penyisipan adalah penambahan sebuah karakter kedalam salah satu susunan karakter dari sebuah string. Contohnya, kata “tepi” ditambahkan huruf ‘s’ dibelakangnya menjadi “tepis”.

2. Penghapusan

   Operasi penghapusan adalah operasi menghapus sebuah karakter dari sebuah string. Contohnya, huruf pertama dari kata “nanak” dihapus dan kata tersebut berubah menjadi “anak”.

3. Penggantian

   Operasi penggantian adalah operasi mengganti sebuah karakter dari sebuah susunan karakter (*string*) menjadi satu karakter lain. Contohnya, penggantian karakter pertama dari kata “gelas” dengan huruf ‘k’ akan mengubahnya menjadi kata “kelas”.

Perlu diperhatikan juga, bahwa selama operasi terhadap karakter kata yang dituju juga tidak harus merupakan kata yang valid. Contohnya, penghapusan karakter pertama dari “celana” menjadi “elana”. Perubahan tersebut diperbolehkan.

Berdasarkan tiga operasi tersebut algoritma ini dapat dibuat dengan fungsi rekursif sebagai berikut:

$$f(m,n)\begin{cases} m, & n=0 \\ n, & m=0 \\ f(m-1,n-1), & str1[i]=str2[j] \\ 1+min\begin{cases} f(m-1,n-1) \\ f(m-1,n) \\ f(m,n-1) \end{cases}, & other \end{cases}$$

Pada fungsi tersebut m menandakan sebuah penghitung untuk string pertama dan n adalah penghitung untuk string kedua. Terdapat empat kasus pada fungsi tersebut, kasus pertama dan kedua adalah ketika salah satu string kosong, kasus kedua adalah ketika string pada elemen tersebut sama, dan kasus keempat mewakili mewakili kasus ketika ada perubahan dan dibutuhkan penggantian, penghapusan, atau penyisipan karakter. Hasil untuk dua string s1 yang memiliki panjang a dan string s2 yang memiliki panjang b bisa didapat dengan melihat nilai pada fungsi f(a,b).

Jika algoritma dibentuk hanya dengan rekursif biasa seperti diatas program akan memiliki kompleksitas waktu sebesar O(3^(mn)). Algoritma akan jauh lebih baik jika dibentuk dengan ide pemrograman dinamis dengan fungsi yang sama sehingga program akan memiliki kompleksitas waktu yang lebih baik yakni O(mn). Lebih jelasnya, berikut contoh tabel pemrograman dinamis untuk string pertama “bejo” dan string kedua “besar”:

m\n	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	0	1	2	3	4
2	2	1	0	1	2	3
3	3	2	1	1	2	3
4	4	3	2	3	2	3

Gambar 2.2.1 contoh tabel dp levensthein

III. Perancangan Solusi

Untuk makalah ini, akan dibuat sebuah program berbahasa python yang menghitung jumlah kata bermakna sama yang cukup mirip dari dua bahasa yang dipilih. Sumber kata akan diambil dari bahasa inggris yang didapat dari tautan pada referensi keempatn lalu diterjemahkan ke kedua bahasa tersebut dengan library googletrans. Untuk setiap kata bermakna sama dari kedua bahasa yang dipilih, kata akan dibandingkan dengan algortima *levenshtein distance*. Pada perancangan ini dua buah kata akan disebut mirip jika jaraknya pada *levenshtein distance* tidak mencapai sepertiga dari jumlah panjang kedua kata.

Pada Implementasinya, program akan terbagi menjadi tiga bagian utama:

A. Pengumpul Kata

Pada bagian ini program akan mengambil kumpulan kata dari sebuah file kemudian memasukkannya ke dalam sebuah array. Berikut Implementasi program pada bagian ini.

```
def get_words():
    res = []
    file = open("english_words.txt", "r")
    for x in file:
        res.append(x)
    return res
```

Gambar 3.1. Implementasi Pengumpul Kata

B. Penerjemah

Seperti namanya, bagian ini akan mengubah kata yang telah dikumpulkan sebelumnya untuk diterjemahkan ke bahasa yang dituju. Berikut implementasinya.

```
def translate_text(text, src_lang, dest_lang):
    translator = Translator()
    try:
        translated = translator.translate(text, src=src_lang, dest=dest_lang)
        if(translated.pronunciation == None):
            return translated.text
        else:
            return translated.pronunciation
    except Exception as e:
        raise e
```

Gambar 3.2. Implementasi Penerjemah

C. Fungsi *Levenshtein Distance*

Fungsi ini akan mengambil dua kata yang didapat dari satu kata yang diterjemahkan ke dua bahasa. Fungsi ini mengembalikan nilai *levenshtein distance* dari kedua kata tersebut. Berikut implementasi fungsi ini.

```
def levenshteinDP(s1, s2):
    m = len(s1)
    n = len(s2)

    dp = [[0 for _ in range(n + 1)] for _ in range(m + 1)]

    # basis
    for i in range(m + 1):
        dp[i][0] = i
    for j in range(n + 1):
        dp[0][j] = j

    # rekurens
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if s1[i - 1] == s2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1]
            else:
                dp[i][j] = 1 + min(dp[i][j - 1], dp[i - 1][j], dp[i - 1][j - 1])
    return dp[m][n]
```

Gambar 3.1. Implementasi Levenshtein Distance

D. Program Utama

Program ini akan menjalankan ketiga program di atas secara berurutan. Mengumpulkan kata, Menerjemahkannya ke dua bahasa, menghitung jarak untuk setiap pasangan kata, dan memberi kelua ran ke layar. Berikut implementasi dari program utama.

```python
if __name__ == "__main__":
    if(str.lower(str(input("Apakah kamu ingin melihat daftar kode bahasa?(Y/N)
    ")))=="y"):
        print(LANGUAGES)

    source_language = "en"
    language_1 = str(input("\nMasukkan kode bahasa pertama:"))
    language_2 = str(input("\nMasukkan kode bahasa kedua:"))


    words = get_words()
    for word in words:
        param = [word, source_language, language_1, language_2]
        pool.apply_async(worker, args=param)

    pool.close()
    pool.join()

    result = []

    for translated_tuple in translated_words:
        distance = levenshteinDP(translated_tuple[1],translated_tuple[2])
        if(distance < (translated_tuple[1].length+translated_tuple[2])/3):
            result.append([translated_tuple[0], translated_tuple[1],
            translated_tuple[2],distance])

    print(f"ditemukan {len(result)} yang mirip yakni:")
    cnt = 1
    for calculated_tuple in result:
        print(f"{cnt}.\nKata dalam bahasa pertama: {calculated_tuple[1]}\nKata
        dalam bahasa kedua: {calculated_tuple[2]}\nJarak: {calculated_tuple[3]}
        ")
        cnt+=1
```

Gambar 3.1. Implementasi Program Utama

IV. Hasil Percobaan

Untuk menjaga keberjalanan dari percobaan ini dan upaya pencegahan pemblokiran google translate, percobaan hanya akan dilakukan pada satu per sepuluh data kata yang ada, menjadi 300 kata dari 3000 kata. Kemudian pada bagian ini juga akan disajikan beberapa hasil percobaan yang dilakukan dengan program yang terlampir.

A. Percobaan pertama

1. Bahasa pertama: Bahasa Indonesia
2. Bahasa kedua: Bahasa Inggris
3. Hasil Percobaan

```
ditemukan 57 yang mirip yakni:
1.
Kata dalam bahasa pertama: klien
Kata dalam bahasa kedua: client
Jarak: 2
2.
Kata dalam bahasa pertama: kolesterol
Kata dalam bahasa kedua: cholesterol
Jarak: 2
3.
Kata dalam bahasa pertama: kolonial
Kata dalam bahasa kedua: colonial
Jarak: 1
4.
Kata dalam bahasa pertama: kode
Kata dalam bahasa kedua: code
Jarak: 1
5.
Kata dalam bahasa pertama: komandan
Kata dalam bahasa kedua: commander
Jarak: 4
6.
Kata dalam bahasa pertama: konferensi
Kata dalam bahasa kedua: conference
Jarak: 3
7.
Kata dalam bahasa pertama: konsentrasi
Kata dalam bahasa kedua: concentration
Jarak: 5
```

Gambar 4.1. Hasil Percobaan Pertama

Pada hasil percobaan pertama didapat 57 kata yang mirip dari 300 kata yang diuji. Ini menunjukkan banyak kata di Bahasa Indonesia yang mirip dengan kata di Bahasa Inggris.

B. Percobaan kedua

1. Bahasa pertama: Bahasa Spanyol
2. Bahasa kedua: Bahasa latin
3. Hasil Percobaan

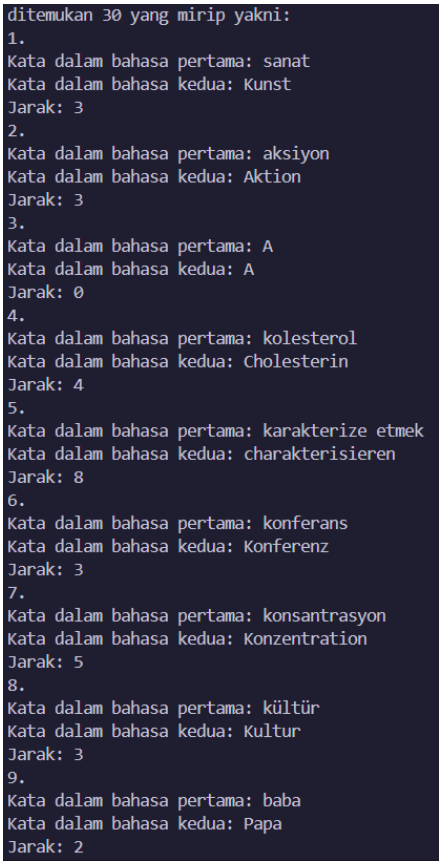
```
ditemukan 104 yang mirip yakni:
1.
Kata dalam bahasa pertama: admisión
Kata dalam bahasa kedua: admissio
Jarak: 3
2.
Kata dalam bahasa pertama: arte
Kata dalam bahasa kedua: artem
Jarak: 1
3.
Kata dalam bahasa pertama: consejo
Kata dalam bahasa kedua: consilio
Jarak: 3
4.
Kata dalam bahasa pertama: absolutamente
Kata dalam bahasa kedua: absolute
Jarak: 5
5.
Kata dalam bahasa pertama: aeronave
Kata dalam bahasa kedua: aircraft
Jarak: 5
6.
Kata dalam bahasa pertama: surgir
Kata dalam bahasa kedua: surge
Jarak: 2
```

Gambar 4.2. Hasil Percobaan Kedua

Pada hasil percobaan kedua, terlihat kemiripan yang cukup besar didapat 104 kata yang mirip dari kurang dari 300 kata yang diuji, dikarenakan terdapat beberapa kata yang gagal ditranslasi. Ini menunjukkan banyak kata di bahasa Spanyol yang mirip dengan kata di Bahasa Latin.

C. Percobaan ketiga

4. Bahasa pertama: Bahasa Turki
5. Bahasa kedua: Bahasa Jerman
6. Hasil Percobaan



Gambar 4.3. Hasil Percobaan Ketiga

Pada percobaan ketiga tidak ditemukan terlalu banyak kata yang mirip antara Bahasa Turki dan Bahasa Jerman. Hal ini dapat terlihat dari kata yang mirip hanya berjumlah 30 dari 300 kata.

V. KESIMPULAN

Makalah ini mengajukan ide penerapan *approximate string matching* untuk mengukur kemiripan antar dua bahasa. Berdasarkan hasil percobaan yang didapat, bisa diperhatikan bahwa beberapa bahasa memiliki kemiripan yang cukup besar dengan bahasa lain sedangkan yang lainnya tidak. Tersebarnya hasil perhitungan bahasa yang mirip juga menunjukkan keberhasilan pendekatan analisis kemiripan bahasa lewat *approximate string matching.* Keberhasilan ini harapannya dapat memicu penelitian lanjutan yang dapat mendukung berkembangnya studi linguistik komparatif.

LINK REPOSITORY GITHUB

https://github.com/RayNoor0/MakalahStima_13522107

UCAPAN TERIMA KASIH

Penulis ingin mengucapkan rasa syukur kepada Tuhan Yang Maha Esa. Karena atas nikmat dan karunia-Nya lah penulis dapat menyelesaikan makalah yang berjudul “Analisis Kemiripan Dua Bahasa dengan Approximate String Matching”. Penulis juga ingin mengucapkan rasa terima kasih kepada tim pengajar mata kuliah Strategi Algoritma, terutama pengajar saya Bapak Dr. Ir. Rinaldi Munir, M.T. untuk pembelarajannya selama satu semester terakhir. Ucapan terima kasih juga penulis sampaikan ke semua pihak yang telah menunjang dalam kelancaran mata kuliah dan penulisan makalah ini.

REFERENSI

[1] https://medium.com/@m.nath/fuzzy-matching-algorithms-81914b1bc498

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Program-Dinamis-2020-Bagian1.pdf

[3] https://www.geeksforgeeks.org/introduction-to-levenshtein-distance/

[4] https://www.ef.com/wwen/english-resources/english-vocabulary/top-3000-words/

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

Ttd

Rayendra Althaf Taraka Noor 13522107