Analisis Kedekatan Struktural dan Leksikal Bahasa-Bahasa Rumpun Austronesia Terhadap Bahasa Indonesia berdasarkan *Regex* dan *Fuzzy Matching*

Muhammad Aufa Farabi - 13523023
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: author@gmail.com , author@std.stei.itb.ac.id

Abstrak*—Bahasa Indonesia adalah bahasa resmi Indonesia dan merupakan salah satu bahasa dengan penutur terbesar di Asia Tenggara. Bahasa ini termasuk dalam Rumpun Bahasa Austronesia bersama dengan bahasa-bahasa seperti bahasa Jawa dan bahasa Tagalog (Filipina). Bahasa-bahasa ini memiliki kemiripan kosakata dan struktur bahasanya yang mencerminkan asal usul yang sama. Kemiripan tersebut menjadi suatu hal yang menarik untuk dianalisis guna memahami evolusi bahasa-bahasa Austronesia, dan mendukung pengembangan teknologi pemrosesan bahasa. Upaya mengukur kedekatan itu, dalam hal ini terhadap bahasa Indonesia, dapat dilakukan dengan mengimplementasi *Regex (Regular Expression)* dan *Fuzzy Matching* berdasarkan algorima *Levehnstein Distance* Pendekatan tersebut diharapkan dapat menghasilkan pengukuran kedekatan leksikal dan Struktural yang sesuai dengan kenyataan linguistik dan memperluas penerapan dari konsep *Regex* dan *Fuzzy Matching.

Kata kunci—Bahasa Indonesia; Rumpun Bahasa Austronesia; Kedekatan Leksikal dan Struktural; Regex; Fuzzy Matching; Algoritma Levehnstein Distance

I. PENDAHULUAN

Bahasa Indonesia adalah bahasa resmi yang dipakai di Indonesia. Bahasa ini merupakan salah satu bahasa dengan jumlah penutur terbesar di dunia, terutama di Asia Tenggara. Dalam hal ini, tidak hanya orang Indonesia saja dengan latar belakang beragam yang disatukan oleh bahasa Indonesia, tetapi juga Negara-negara di sekitar yang memiliki kemampuan untuk memahami atau berbahasa bahasa Indonesia, seperti Malaysia dan Timor Leste.

Bahasa Indonesia sendiri tergolong dalam Rumpun bahasa Austronesia, yakni suatu rumpun bahasa yang mencakup ribuan bahasa yang tersebar luas di Asia Tenggara dan Kepulauan Pasifik. Bahasa-bahasa pada rumpun bahasa Austronesia ini memiliki sejarah asal usul yang sama dan memiliki kemiripan pada kosakata dan struktur bahasanya, contohnya terjemahan kata “aku” pada bahasa Tagalog adalah “ako”. Kemiripan kosakata dan struktur bahasa dari bahasa-bahasa tersebut terutama dengan bahasa Indonesia merupakan suatu topik yang menarik karena hubungan tersebut dapat mengungkap bagaimana asal-usul dan evolusi dari bahasa Indonesia, dan lebih penting lagi untuk tujuan pengembangan bahasa, seperti teknologi pemrosesan bahasa alami (NLP) atau pengembangan kamus bahasa-bahasa Austronesia terhadap bahasa Indonesia.

Upaya memahami suatu kedekatan leksikal dan struktural dari berbagai bahasa pada Rumpun Bahasa Austronesia tidaklah mudah karena perkembangan setiap bahasa yang sudah berlangsung lama telah menyebabkan perbedaan yang sangat besar satu sama lain, ditambah lagi cakupan geografis Rumpun Bahasa Austronesia yang sangat luas sehingga terdapat kontak atau pengaruh dari besar bahasa-bahasa non-Austronesia. Walaupun dengan adanya perbedaan yang besar, setiap bahasa Austronesia memiliki suatu kemiripan yang khas yang membedakan dengan bahasa-bahasa non Austronesia, seperti kosakata dasar yang serumpun, pemakaian reduplikasi dan imbuhan, dan jumlah fonem yang relatif sedikit. Untuk itu, diperlukan suatu metode atau teknik yang dapat menganalisis kedekatan struktur bahasa dan leksikal untuk bahasa-bahasa dari Rumpun Bahasa Austronesia, dalam hal ini komparasi dengan bahasa Indonesia secara *general* dan efisien.

Di sini, konsep algoritma String Matching, dalam bentuk Fuzzy Matching bersama dengan penerapan *Regular Expression* dapat digunakan untuk mengungkap ukuran kedekatan bahasa-bahasa Austronesia terhadap bahasa Indonesia dalam hal struktural dan leksikal. Penggunaan *regular expression* ditujukan untuk mengidentifikasi struktur bahasa dari suatu bahasa. Kemudian, kata-kata yang serumpun dari bahasa-bahasa yang dibandingkan dapat diukur kemiripannya kosakatanya dengan *Fuzzy Matching*, dengan algoritma yang dipakai adalah algoritma *Levehnstein Distance* yang dapat mengukur kemiripan suatu kata dengan akurat. Hasil dari pemrosesan ini menjadi suatu analisis ukuran kedekatan struktural dan leksikal antara bahasa-bahasa Austronesia terhadap bahasa Indonesia dan efektivitas dari pemakaian *Regular Expression* dan *Fuzzy Matching* dalam pemrosesan perhitungan pengukurannya sebagai topik utama dari makalah ini.

II. LANDASAN TEORI

A. *Rumpun Bahasa*

Rumpun bahasa adalah kumpulan bahasa-bahasa yang memiliki asal usul atau nenek moyang bahasa yang sama, yakni disebut proto-bahasa. Bahasa-bahasa dalam suatu rumpun bahasa memiliki keterkaitan dalam kesamaan kosakata, struktur bahasa, dan fonologinya. Terdapat beberapa rumpun-rumpun bahasa besar yang bahasa-bahasanya dituturkan oleh mayoritas penduduk dunia, yaitu rumpun bahasa Indo-Eropa, Sino-Tibetan, Afroasiatik, dan Austronesia. Bahasa-bahasa pada suatu rumpun bahasa seiring waktu akan menjadi semakin berbeda yang disebabkan oleh faktor geografis, kultural, dan kontak dengan bahasa lain yang pada akhirnya akan membentuk suatu kumpulan cabang bahasa-bahasa itu sendiri.

B. *Rumpun Bahasa Austronesia*

Salah satu rumpun bahasa terbesar di dunia adalah rumpun Bahasa Austronesia. Rumpun bahasa ini merupakan suatu rumpun bahasa yang terdiri dari 1200 bahasa, termasuk bahasa Indonesia, dan memiliki cakupan geografis yang sangat luas di dunia, yakni dari pulau Madagascar hingga Kepulauan Pasifik. Rumpun bahasa Austronesia diyakini berasal dari suatu bahasa nenek moyang yang dituturkan di pulau Taiwan sekitar 5000 tahun yang lalu yang kemudian berevolusi dan menyebar ke Asia Tenggara dan Kepulauan Pasifik oleh migrasi laut Bangsa Austronesia. Beberapa contoh bahasa dengan banyak penutur yang termasuk dalam rumpun Bahasa Austronesia adalah bahasa Indonesia, bahasa Jawa, bahasa Filipino, dan bahasa Maori.

Gambar 1. Peta Persebaran Rumpun Bahasa Austronesia (Sumber: https://www.languagesgulper.com/eng/Austronesian.html)

Pada dasarnya, Rumpun Bahasa Austronesia terbagi menjadi dua cabang keluarga, yakni bahasa-bahasa Formosa yang ditutur di pulau Taiwan dan cabang Melayu-Polinesia yang mencakup semua bahasa selain itu. Kelompok bahasa Melayu-polinesai ini juga dibagi menjadi dua sub-cabang utama, yaitu Melayu-Polinesia barat yang mencakup bahasa-bahasa di Indonesia Barat dan Filipina serta Melayu-Polinesia tengah-timur yang mencakup bahasa-bahasa di Indonesia Timur dan utamanya di Kepulauan Pasifik.

Bahasa-bahasa Rumpun Austronesia memiliki kemiripan yang ditandai pada struktur bahasa dan kosakatanya. Secara fonologi (bunyi dari bahasa), bahasa-bahasa ini umumnya hanya tidak memiliki jumlah huruf konsonan yang besar dan kebanyakan kata nya mengandung huruf konsonan dan vocal yang berselang-seling, contohnya "**m** a **k** a **n**". Hal yang sering dijumpai pada struktur bahasa rumpun Bahasa Austronesia adalah pemakaian imbuhan, seperti *n-gugah* (mem-bangun-kan) dalam bahasa *gugah* dan juga reduplikasi kata yang salah satunya bersifat untuk menunjukkan suatu hal jamak, contohnya anak-anak (beberapa anak). Kosakata yang serumpun pada kebanyakan bahasa dari Rumpun Bahasa Austronesia umumnya berkaitan dengan konsep-konsep dasar atau sandang pangan.

Berikut adalah contoh kemiripan kosakata bahasa-bahasa dari Rumpun Bahasa Austronesia untuk angka 1-10.

Austronesian Shared Vocabulary

Language	one	two	three	four	five	six	seven	eight	nine	ten
Samoan	tasi	lua	tolu	fa	lima	ono	fitu	valu	iva	sefulu
Rarotongan	tai	rua	toru	a	rima	ono	itu	varu	iva	tai'ngauru
Hawaiian	kahi	lua	kolu	ha	lima	ono	hiku	walu	iwa	'umi
Tuvalu	tasi	iua	tolu	fa	lima	ono	fitu	valu	iva	sefulu
Rapanui	tahi	rua	toru	ha	rima	ono	hita	vau	iva	angahuru
Rotuman	ta	rua	folu	hake	lima	ono	hifu	valu	siva	saghula
Maori	tahi	rua	toru	wha	rima	ono	whitu	waru	iwa	tekau
Tongan	taha	ua	tolu	fa	nima	ono	fitu	valu	hiva	hongofulu
Fijian	dua	rua	tolu	va	lima	ono	vitu	walu	civa	sagavalu
Ceuano (Visayan) Philipines	usa	duha	tulo	upat	lima	unum	pito	walo	siyam	napulo
Tagalog	isa	dalawa	tatlo	apat	lima	anim	pito	walo	siyam	sampu
Bahasa Malaysia	satu	dua	tiga	empat	lima	enam	tujuh	lapan	sembila	sepuluh

Gambar 2. Perbandingan kosakata angka pada Rumpun Bahasa Austronesia (Sumber: https://www.languagesoftheworld.info/oceania-and-madagascar/speak-hawaiian.html)

Perbedaan lain dari struktur dari bahasa-bahasa Rumpun Bahasa Austronesia terletak pada *word order* atau urutan katanya. Mayoritas bahasa di Filipina dan Taiwan memakai *word order* VSO atau VOS, dengan V = kata kerja, S = kata subjek, dan O = kata objek/benda, sedangkan bahasa-bahasa di Indonesia dan Kepulauan Pasifik menggunakan *word order* SVO atau yang lebih dikenal sebagai S-P-O-K.

C. *String Matching*

String Matching adalah suatu proses untuk menemukan keberadaan sebuah *string* (berupa kata atau frasa) yang bersesuaian dengan suatu *pattern* dalam teks dengan panjang karakter yang lebih besar. Contoh dasar dari *String Matching* adalah seperti berikut

```
T: the rain in spain stays mainly on the plain
P: main
```

Gambar 3. Contoh Dasar String Matching (Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf)

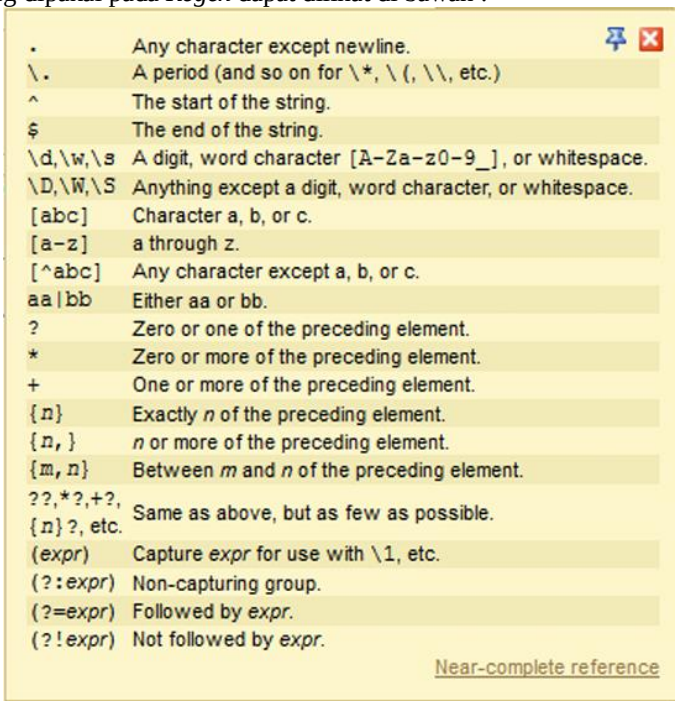
Terdapat banyak aplikasi atau implementasi yang bias diterapkan dari *String Matching*, seperti untuk mendeteksi plagiarisme, mengecek urutan DNA, penyaringan konten internet, dan lain-lain.

String Matching dibagi menjadi dua dalam hal algoritmanya, yakni

- Exact String Matching (**Exact Matching**), menemukan keberadaan *string* dalam teks yang persis atau eksak dengan *pattern*
- Approximate String Matching (**Fuzzy Matching**), menemukan *string* yang berkesesuaian dengan *pattern* secara aproksimasi

D. *Regular Expression*

Regular Expression (Regex) adalah suatu teknik yang mendefinisikan suatu pola pencarian (*string*) yang digunakan dalam pencocokkan suatu substring dalam suatu teks dengan pola yang ditentukan yang selanjutnya *substring* tersebut dapat ditemukan, diekstrak, atau dimodifikasi. Suatu pola *Regex* terdiri atas serangkaian karakter literal, karakter khusus, dan operator yang memiliki makna tertentu dalam proses pencocokkan *string*-nya. Notasi-notasi karakter atau operator yang dipakai pada *Regex* dapat dilihat di bawah :

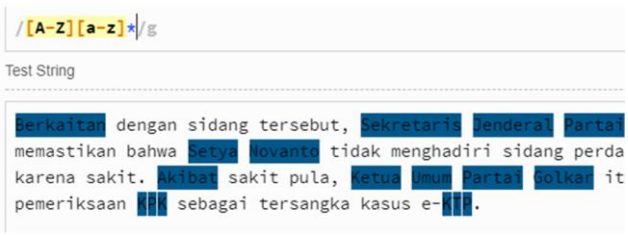


.	Any character except newline.
\.	A period (and so on for *, \(, \\, etc.)
^	The start of the string.
$	The end of the string.
\d,\w,\s	A digit, word character [A-Za-z0-9_], or whitespace.
\D,\W,\S	Anything except a digit, word character, or whitespace.
[abc]	Character a, b, or c.
[a-z]	a through z.
[^abc]	Any character except a, b, or c.
aa|bb	Either aa or bb.
?	Zero or one of the preceding element.
*	Zero or more of the preceding element.
+	One or more of the preceding element.
{n}	Exactly *n* of the preceding element.
{n,}	*n* or more of the preceding element.
{m,n}	Between *m* and *n* of the preceding element.
??,*?,+?, {n}?, etc.	Same as above, but as few as possible.
(*expr*)	Capture *expr* for use with \1, etc.
(?:*expr*)	Non-capturing group.
(?=*expr*)	Followed by *expr*.
(?!*expr*)	Not followed by *expr*.

Near-complete reference

Gambar 4. Notasi umum Regex dan Kegunaannya (Sumber: https://www.languagesgulper.com/eng/Austronesian.html)

Pada gambar di atas, bisa dilihat bahwa regex memiliki notasi-notasi yang dapat digunakan mencocokkan suatu *substring* dalam teks dengan pola yang terbentuk dari hasil kombinasi pemakaian notasi-notasinya. Contoh pemakaian notasi-notasi *Regex* ini adalah mencari suatu kata yang diawali dengan huruf kapital (huruf besar) yang dilanjutkan dengan nol, satu atau lebih karakter huruf kecil.



Gambar 5. Contoh Regex (Sumber: https://www.languagesgulper.com/eng/Austronesian.html)

Di sini, notasi [A-Z][a-z]* mendefinisikan suatu pola kata yang ingin dicari dengan alphabet besar sebagai kemunculan huruf pertama ([A-Z]) yang kemudian dilanjutkan dengan nol atau banyak (*) huruf kecil ([a-z]). Pola yang dibentuk dari notasi tersebut akan dipakai oleh mesin *Regex* untuk dicocokkan dengan *substring* kata yang berkesuaian dalam teks yang diinput dan kemudian menghasilkan hasil *substring* yang cocok atau posisi *substring* yang ditemukan.

Pemakaian *Regex* yang relatif simpel, yakni lebih ditekankan pada kombinasi pemakaian notasi dibandingkan pembuatan algoritma, menjadikan *Regex* sebagai alat yang penting dalam String Matching, khususnya untuk Exact Matching. Di sisi lain, Regex dapat melakukan pencocokkan kata secara efisien dan sudah terstandarisasi pada berbagai platform.

E. *Fuzzy Matching*

Fuzzy Matching, atau lebih tepatnya Approximate String Matching, adalah suatu jenis String Matching yang mengidentifikasi string dalam teks yang memiliki kemiripan dengan suatu *pattern*, tetapi tidak identical. *Fuzzy Matching* memiliki kelebihan dibandingkan Exact Matching dalam mencocokkan suatu kata dalam teks yang tidak harus eksak atau terdapat typo di dalamnya. Dalam arti lain, Fuzzy Matching memberikan fleksibilitas dalam mencocokkan suatu string berdasarkan seberapa mirip karakter dan urutan karakternya. Hal ini membuat Fuzzy Matching dapat diimplementasikan dalam pencocokkan suatu string dalam teks yang mungkin memiliki format yang berbeda-beda.

Terdapat beberapa algoritma Fuzzy Matching yang digunakan untuk mengukur kemiripan dari satu atau beberapa string dengan proses yang berbeda. Algoritma tersebut di antaranya adalah

- Algoritma Levehnstein Distance
- Algoritma Hamming Distance
- Algoritma Bitmap

Pada makalah ini, algoritma Fuzzy Matching yang dipakai adalah algoritma Levehnstein Distance. Algoritma tersebut dalam implementasinya akan memakai *dynamic programming* untuk

F. *Dynamic Programming*

Dynamic Programming (DP) adalah metode pemecahan suatu masalah kompleks dengan membaginya menjadi submasalah yang lebih kecil yang dikerjakan pada suatu tahap. Submasalah-submasalah itu dipastikan hanya menghitung hasil solusi optimalnya sekali dalam tahap nya yang kemudian disimpan untuk penggunaan pada tahap penyelesaian masalah selanjutnya. Karena itu, *Dynamic programming* dapat digunakan untuk menyelesaikan suatu persoalan secara optimal dengan efisien karena setiap keputusan dari tahap atau solusi dari sub-masalah diperhitungkan dalam pencarian solusi optimal yang global.

Pada *Dynamic programming*, dikenal suatu istilah yang bernama prinsip optimalitas, yaitu rangkaian keputusan yang optimal. Prinsip tersebut menyatakan bahwa jika solusi total optimal maka bagian solusi dari tiap tahap pencarian solusi juga optimal.

Ada dua cara untuk menyimpan hasil solusi optimal dari setiap submasalah, yakni secara *top-down* (*Memoization*) dan secara *bottom-up.*

G. *Algoritma Levehnstein Distance*

Konsep algoritma Levehnstein distance tidak bisa dipisahkan dari definisi Levehnstein Distance itu sendiri. *Levehnstein Distance* atau *edit Distance* adalah suatu ukuran kemiripan antara dua string yang dihitung berdasarkan jumalh operasi minimal yang dilakukan untuk mengubah suatu string menjadi string lain. Operasi yang dimaksud antara lain

- Substitusi, mengganti satu karakter dengan karakter lain
- *Insert* (penyisipan), Menambahkan satu karakter ke string
- *Delete* (penghapusan), Menghapus satu karakter dari string

Jadi, algoritma *Levehnstein Distance* merupakan algoritma yang mengukur seberapa beda suatu string dengan string lain.

Persamaan yang dipakai pada algoritma *Levehnstein Distance* adalah seperti berikut:

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Gambar 6. Persamaan Algoritma *Levehnstein Distance*
(Sumber: https://medium.com/@ethannam/understanding-the-levenshtein-distance-equation-for-beginners-c4285a5604f0)

Dengan a sebagai string pertama, b sebagai string kedua, i adalah posisi terminal karakter pada string pertama, dan j adalah posisi terminal karakter pada string kedua.

Terdapat beberapa cara dalam menerapkan algoritma *Levehnstein Distance*, tetapi cara yang paling efisien dalam implementasi algoritma *Levehnstein Distance* adalah dengan memakai *dynamic programming*. Hal ini dilakukan dengan mengisi sebuah matriks DP sebesar (m+1) x (n+1) dengan m adalah panjang string a dan n adalah panjang string b. Setiap cell pada matriks DP ini menyimpan jarak minimum dari a[0…i-1] dan b[0….i-1]

Penerapan dari Algoritma *Levehnstein Distance* dapat diperlihatkan untuk kasus menghitung *edit distance* dari string “kota” dengan string “kita”. Panjang "kota" adalah 4 dan "kita" juga 4 sehingga tabel matriks DP berukuran (4+1) x (4+1) atau 5×5. Setiap sel dp[i][j] menyatakan jarak minimum untuk mengubah kota[0...i-1] menjadi kita[0...j-1]. Langkah awal dari perhitungannya adalah mengisi baris pertama dengan dp[0][j] = j dan kolom pertama dengan dp[i][0] = I karena mengubah string kosong menjadi string sepanjang j memerlukan sebanyak j operasi penyisipan. Setelah itu, cell-cell pada matriks DP diisi dengan rumus persamaan Algoritma *Levehnstein Distance* yang sudah dijelaskan sebelumnya.

Berikut adalah tabel Matriks DP yang dihasilkan:

	""	k	i	t	a
""	0	1	2	3	4
k	1	0	1	2	3
o	2	1	1	2	3
t	3	2	2	1	2
a	4	3	3	2	1

Dari tabel di atas, terlihat bahwa dp[4][4] = 1 yang menyimbolkan hasil *levehnstein Distance* yang dicari, artinya hanya satu operasi (substitusi huruf 'o' dengan 'i') yang dibutuhkan untuk mengubah "kota" menjadi "kita".

III. Implementasi dan Pengujian

A. *Implementasi*

Untuk melakukan implementasi dari konsep *Regex* dan Fuzzy Matching dalam menganalisis kedekatan leksikal dan *structural* dari bahasa-bahasa Austronesia terhadap bahasa Indonesia, dilakukan penentuan bahasa-bahasa yang akan dipilih untuk pengujian terhadap bahasa Indonesia. Di sini, bahasa yang dipilih ditentukan dari kedekatan keluarga bahasa mulai yang paling dekat hingga jauh dari bahasa Indonesia. Bahasa-bahasa yang dipilih tersebut adalah

- Bahasa Melayu
- Bahasa Aceh
- Bahasa Tagalog

Bahasa-bahasa tersebut bersama dengan bahasa Indonesia akan dibuatkan beberapa kalimat yang berkesesuaian (sama makna) yang akan dibandingkan masing-masingnya dengan pemrosesan *Regex* dan *Fuzzy Matching* (algoritma *Levehnstein Distance*). Hasil yang akan dicapai disini adalah ukuran persentase seberapa mirip ketiga bahasa tersebut dengan bahasa Indonesia dalam hal kosakata dan struktural kalimat nya, dengan keterangan struktur yang akan ditinjau adalah pemakaian reduplikasi mengingat hal tersebut merupakan ciri

khas dari rumpun Bahasa Austronesia serta penerapan *Regex* dan *Fuzzy Matching* secara murni tidak cukup untuk menganalisis struktur yang lebih kompleks, seperti *word order* atau imbuhan, mengingat rumpun Bahasa Austronesia yang sangat beragam.

Program yang diimplementasikan menggunakan bahasa Python atas alasan *syntax* yang mudah dibaca serta *library* bahasa python yang lengkap. Di sini, sampel data yang akan dipakai untuk komparasi kemiripan kosakata dan struktur reduplikasi dibuat dalam file txt dengan format isi nya berupa [KodeBahasa], [Kalimat]. Sampel data tersebut dibaca ke dalam program sebagai suatu dictionary dengan *key* nya adalah kode bahasa dan *value* nya adalah list dari sampel data kalimat untuk bahasa tersebut. Di bawah ini adalah fungsi untuk memproses sampel data untuk program:

```python
# Baca file txt dan kelompokkan kalimat berdasarkan bahasa sebagai dict
def readFromFile(filename: str) -> Dict[str, List[str]]:
    groups = defaultdict(list)
    with open(filename, 'r', encoding='utf-8') as f:
        for line in f:
            if ',' not in line:
                continue
            # buat pemetaan kode bahasa dengan kalimatnya
            code, sentence = line.strip().split(',', 1)
            groups[code.strip()].append(sentence.strip())
    return groups
```

Gambar 7. Fungsi readFromFile(Sumber:Dokumen Pribadi)

Fungsi di atas bekerja dengan membuat suatu dictionary yang menyimpan kode bahasa dan list dari kalimat yang bersesuaian dengan bahasa tersebut. Fungsi ini membaca file .txt dari baris ke baris yang kemudian baris tersebut dipisah menjadi kode bahasa (*code*) dan kalimat bahasanya dengan separator “,”. Kalimat bahasa yang didapat dari pemisahan tersebut akan dimasukkan ke dalam dictionary dengan key yang berasal dari kode bahasa tersebut.

Setelah didapatkan *dictionary* untuk kalimat-kalimat bahasa, *dictionary* tersebut diproses dalam suatu fungsi untuk menghitung ukuran kemiripan berdasarkan leksikal dan struktur bahasanya. Setiap kalimat dari tiap bahasa akan dibandingkan dengan kalimat dari bahasa Indonesia untuk index kalimat yang berkesusuaian dengan memanggil fungsi penghitungan kemiripan leksikal dan kemiripan structural (berdasarkan reduplikasi). Hasil perhitungan untuk tiap kalimat dalam leksikal dan structuralnya disimpan dalam suatu dictionary dengan key nya adalah kode bahasa yang dibandingkan. Gambaran lebih detail mengenai pemrosesan ini dapat dilihat pada gambar di bawah ini

```python
# Analisis kesamaan dengan hanya reduplikasi
def analyzeSimiliarity(langSentencesDict: Dict[str, List[str]], referenceLang='ID'):
    results = {}
    refLangSentences = langSentencesDict.get(referenceLang)
    if not refLangSentences:
        raise ValueError(f"Tidak ada kalimat untuk bahasa referensi '{referenceLang}'")

    for code, sentences in langSentencesDict.items():
        # abaikan bahasa referensi
        if code == referenceLang:
            continue
        # dictionary untuk menyimpan hasil ukuran kemiripan dari tiap kalimat per satu bahasa
        sentencesResult = []
        for i in range(len(refLangSentences)):
            # hitung ukuran kemiripan leksikal dan struktural untuk tiap kalimat berkesuaian dari setiap bahasa dengan bahasa Indonesia
            lexical = lexicalSimiliarity(refLangSentences[i], sentences[i])
            structural = reduplicationSimiliarity(refLangSentences[i], sentences[i])
            sentencesResult.append({
                "Lexical": round(lexical, 3),
                "Structural": round(structural, 3),
            })
        results[code] = sentencesResult
    return results
```

Gambar 8. Fungsi analyzeSimiliarity(Sumber: Dokumen Pribadi)

Perhitungan ukuran kemiripan leksikal dari kalimat yang berkesuaian antar bahasa dilakukan dengan implementasi algoritma *Levehnstein Distance* yang akan menghasilkan jarak Levehnstein dari dua string (kalimat) yang dikomparasi.

```python
def levehnsteinDistance(s1, s2):
    m, n = len(s1), len(s2)
    dp = [[0] * (n + 1) for _ in range(m + 1)]

    # Inisialisasi Baris dan Kolom Pertama
    for i in range(m + 1):
        dp[i][0] = i
    for j in range(n + 1):
        dp[0][j] = j

    for i in range(1, m + 1):
        for j in range(1, n + 1):
            cost = 0 if s1[i-1] == s2[j-1] else 1
            dp[i][j] = min(
                dp[i - 1][j] + 1,        # Deletion
                dp[i][j - 1] + 1,        # Insertion
                dp[i - 1][j - 1] + cost  # Substitution
            )
    return dp[m][n]
```

Gambar 9. Algoritma Levehnstein Distance (Sumber: Dokumen Pribadi)

Hasil dari perhitungan algoritma *Levehnstein Distance* tersebut diubah menjadi suatu skala kemiripan leksikal berdasarkan *range* 0-1, dengan 1 menyimbolkan kemiripan yang identik, sedangkan 0 menyimbolkan tidak ada kemiripan. Hal ini dilakukan pada fungsi lexicalSimiliarity yang dipanggil pada fungsi levehnsteinDistance.

```python
# Menghitung ukuran kemiripan leksikal antar dua kalimat dengan levehnstein distance
def lexicalSimiliarity(sent1: str, sent2: str) -> float:
    max_len = max(len(sent1), len(sent2))
    if max_len == 0:
        return 1.0  # identik
    dist = levehnsteinDistance(sent1, sent2)
    return 1 - dist / max_len
```

Gambar 10. Fungsi perhitungan Lexical Similiarity (Sumber: Dokumen Pribadi)

Terkait dengan perhitungan ukuran kemiripan struktural antar kalimat bersesuaian dari bahasa yang berbeda, dilakukan implementasi Regex untuk mengecek tipe reduplikasi apa dari suatu kata dalam kedua kalimat yang dibandingkan. *Regex* pada program diimplementasikan dari impor modul re untuk bahasa Python.

```python
# cek tipe reduplikasi dari kata
def isReduplicate(word: str) -> str:
    # kasus kata berulang (anak-anak)
    if re.fullmatch(r'(\w+)-\1', word):
        return "pure"
    elif re.fullmatch(r'\w+-\w+', word):
        part1, part2 = word.split('-')
        # ada imbuhan awal
        if part1[1:] == part2:
            return "derived_first"
        # ada imbuhan akhir
        elif part1 == part2[:-1]:
            return "derived_last"
        # ada imbuhan awal dan akhir
        elif part1[1:] == part2[:-1]:
            return "derived_first_last"
        else:
            return "none"
    else:
        return "none"
```

Gambar 11. Fungsi isReduplicate (Sumber: Dokumen Pribadi)

Ada dua pattern *Regex* yang dipakai pada kode tersebut, yaitu

- (\w+)-\1,

  Artinya mencocokkan suatu kata dengan menggunakan sebuah group `()` yang menangkap satu atau lebih karakter huruf/angka/underscore (`\w+`) dengan karakter selanjutnya harus berupa tanda hubung "-" dan bagian setelahnya harus sama dengan isi group pertama (`\1`, yaitu referensi ke grup sebelumnya). Contoh kata yang cocok dengan pattern regex ini ialah "anak-anak", "kura-kura" karena bagian sebelum dan sesudah tanda hubung identik.

- \w+-\w+

  Artinya mencocokkan dua bagian kata yang masing-masing terdiri dari satu atau lebih karakter huruf/angka/underscore (\w+) yang dipisahkan oleh tanda hubung "-". Pola *regex* ini mencakup semua bentuk kata berulang yang memiliki format dua bagian yang dipisah tanda hubung, baik identik maupun tidak. Contoh kata yang cocok adalah "anak-anak" (tidak termasuk dalam kasus ini karena sudah dihandle di kondisi regex sebelumnya), "anak-kanak", "berlari-larian", dan "sayur-mayur".

Pada Regex kedua, terdapat kemungkinan bahwa kata yang memiliki dua bagian yang dipisah oleh tanda hubung tidak identik sama sekali, seperti "mobil-besar". Oleh karena itu, dilakukan pengecekan kondisional tambahan pada kata yang cocok dengan pattern Regex kedua untuk mengecek apakah kata tersebut memang berbeda dari imbuhan awal dan/atau akhir atau memang tidak identik sama sekali. Fungsi isReduplicate di sini akan mengembalikan tipe-tipe reduplikasi dalam bentuk string.

```python
def reduplicationSimiliarity(sentence1: str, sentence2: str) -> float:
    # filter kata yang berulang dari kalimat
    def extractReduplicatedWords(sent):
        words = sent.lower().split()
        return [
            (word, isReduplicate(word.strip(string.punctuation)))
            for word in words
            if isReduplicate(word.strip(string.punctuation)) != "none"
        ]

    rWord1 = extractReduplicatedWords(sentence1)
    rWord2 = extractReduplicatedWords(sentence2)

    min_len = min(len(rWord1), len(rWord2))
    if min_len == 0:
        return 1.0 if len(rWord1) == len(rWord2) else 0.0

    match = 0
    for i in range(min_len):
        _, type1 = rWord1[i]
        _, type2 = rWord2[i]
        if type1 == type2:
            match += 1

    return match / max(len(rWord1), len(rWord2))
```

Gambar 12. Fungsi reduplicationSimiliarity (Sumber: Dokumen Pribadi)

Fungsi isReduplicate sebelumnya dipanggil pada Fungsi reduplicationSimiliarity yang bertujuan untuk mengukur kemiripan struktural dari dua kalimat dari bahasa yang berbeda. Masing-masing kalimat akan diekstrak kata yang bersifat reduplikasi kemudian dibandingkan untuk satu per satu kata untuk mengecek apakah tipe reduplikasi pada posisi ke-I kalimat yang difilter sama atau tidak. Hasil pengukuran ini dihitung dengan pembagian total kecocokan tipe reduplikasi dibagi dengan maksimal dari jumlah reduplikasi dari kedua kalimat.

B. *Pengujian dan Hasil Analisis*

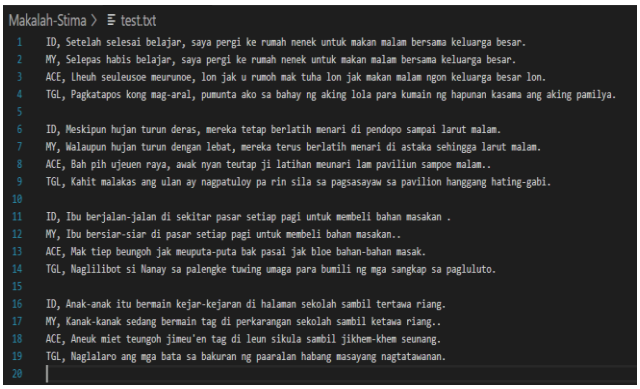
```python
if __name__ == "__main__":
    filename = input("Masukkan nama file teks (dalam .txt): ").strip()
    try:
        langSentencesDict = readFromFile(filename)
        results = analyzeSimiliarity(langSentencesDict, referenceLang="ID")

        for lang in results:
            print(f"\nHasil untuk {lang}:")
            for i, entry in enumerate(results[lang]):
                print(f"Kalimat {i+1}: {entry}")
    except FileNotFoundError:
        print(f"[ERROR] File '{filename}' tidak ditemukan.")
    except Exception as e:
        print(f"[ERROR] Terjadi kesalahan: {e}")
```

Gambar 13. Main Program (Sumber: Dokumen Pribadi)

Program pengukuran kedekatan leksikal dan struktural yang dianalisis dieksekusi pada main function python seperti pada gambar di atas. Program akan meminta input file txt berisi konfigurasi kode bahasa dengan kalimatnya dengan format [KodeBahasa], [Kalimat] yang dipastikan memiliki padanan pada bahasa lain yang juga ingin diuji. Setelah itu, program akan membaca file txt tersebut dengan hasilnya adalah dictionary berisi Kode bahasa dan list kalimat yang sesuai

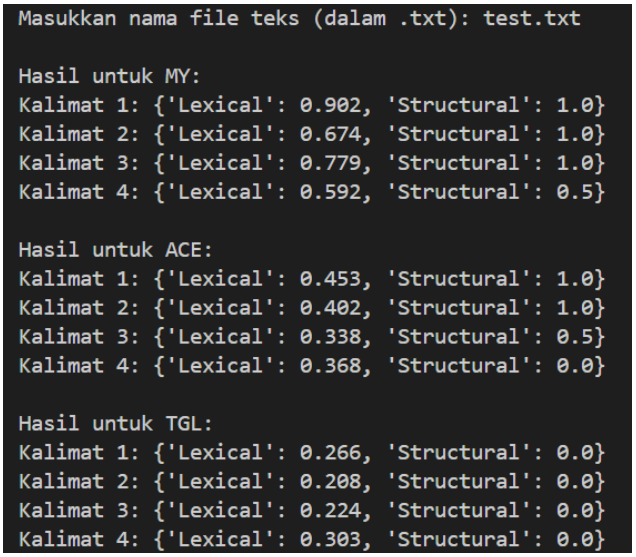
dengan bahasanya. Selanjutnya, program akan memproses hasil dictionary tersebut untuk mengukur kemiripan leksikal dan struktural dari setiap kalimat yang berkesuaian untuk tiap bahasa yang menghasilkan suatu dictionary result berisi kode bahasa dengan hasil ukuran kedekatan leksikal dan struktural dari tiap kalimat pada bahasanya. Hasil result ini ditampilkan ke dalam layar terminal berdasarkan setiap bahasa yang dibandingkan dengan bahasa Indonesia.

Berikut adalah gambaran file txt yang diujikan pada analisis makalah ini:



Gambar 14. Teks Pengujian dalam file .txt
(Sumber:Dokumen Pribadi)

File txt yang digunakan dalam pengujian ini mengikuti konfigurasi yang diperbolehkan untuk pemrosesan teksnya, yakni [KodeBahasa]-[Kalimat]. Terdapat empat buah kalimat dari masing-masing empat bahasa dari Rumpun Bahasa Austronesia dengan salah satunya sebagai titik referensi, yaitu bahasa Indonesia, yang bersesuaian secara makna untuk setiap urutan kalimat (kalimat 1 bahasa Indonesia memiliki makna yang sama dengan kalimat 1 bahasa Aceh). Dua kalimat pertama akan menguji kemiripan dasar dalam kosakata, sedangkan dua kalimat terakhir akan menguji kemiripan struktur reduplikasi kata selain dari kemiripan kosakatanya.



Gambar 15. Hasil Pengukuran Kedekatan Leksikal dan Struktural (Reduplikasi) (Sumber:Dokumen Pribadi)

Berdasarkan hasil ukuran kemiripan leksikal dan struktural reduplikasi kata dari bahasa-bahasa yang dibandingkan dengan bahasa Indonesia, dapat diketahui dengan jelas bahwa terdapat perbedaan tingkat kemiripan baik secara leksikal maupun struktural dari bahasa Melayu, bahasa Aceh, dan bahasa Tagalog. Di sini, bahasa Melayu memiliki tingkat kedekatan leksikal dan struktural yang tertinggi secara rata-rata di antara kedua bahasa lainnya. Hal itu bisa dibuktikan karena bahasa Indonesia memang berasal dari bahasa Melayu dan perbedaan antara bahasa Indonesia dengan bahasa Melayu muncul kurang dari 100 tahun yang lalu. Bahasa Aceh menempati peringkat kedua untuk kedekatan leksikal dan struktural secara rata-rata, walaupun persentase kedekatan leksikalnya cukup turun jauh dibandingkan dengan bahasa Melayu. Walaupun bahasa Aceh merupakan kerabat dari bahasa Melayu, bahasa ini sudah terpisah dengan keluarga bahasa Melayu (seperti bahasa Indonesia, bahasa Melayu, bahasa Minangkabau, dan lain-lain) sejak 2000 tahun yang lalu dan mendapat pengaruh besar dari bahasa-bahasa lain sehingga terjadi perubahan besar dalam kosakatanya. Bahasa terakhir, yakni bahasa Filipino, menempati peringkat terakhir dari perbandingan kedekatan analisis leksikal dan struktural terhadap bahasa Indonesia yang bisa dijelaskan karena kekerabatan yang cukup jauh dengan bahasa Indonesia dibandingkan bahasa-bahasa sebelumnya.

Pengujian ini berhasil membuktikan bahwa *Regular Expression* (Regex) dan *Fuzzy Matching* dengan algoritma *Levehnstein Distance* dapat menghasilkan hasil ukuran kedekatan leksikal dan struktural yang bisa dibuktikan dari linguistik bahasa itu sendiri. *Regex* dapat secara efektif membantu identifikasi dari pola reduplikasi sebagai salah satu struktural bahasanya, sayangnya komponen-komponen lain dari struktural bahasa seperti *word order* atau imbuhan belum dapat diidentifikasi dengan baik oleh pemakaian *Regex* secara murni karena keterbatasannya. Algoritma *Levehnstein Distance* yang dipakai pada program ini dipastikan menghasilkan solusi yang optimal untuk hasil pengukuran kedekatan leksikal karena berbasis *dynamic programming*. Pemakaian *dynamic programming* untuk algoritma *Levehnstein Distance* menghasilkan kompleksitas waktu yang cukup baik dalam pemrosesan programnya, yakni O(m x n) dengan m adalah panjang string 1 dan n adalah panjang string 2. Walaupun begitu, masih ada ruang untuk mengoptimasi algoritma *Levehnstein Distance* terutama untuk kasus spesifik.

IV. KESIMPULAN

Penerapan *Regex* (*Regular Expression*) dan *Fuzzy Matching* yang menggunakan algoritma *Levehnstein Distance* telah berhasil menghasilkan suatu analisis kedekatan struktural dan leksikal suatu bahasa dari Rumpun Bahasa Austronesia terhadap Bahasa Indonesia dalam cakupan tertentu. Meskipun demikian, terdapat keterbatasan dalam hal yang bisa diukur terutama dalam struktur bahasa dengan implementasi murni *Regex* dan *Fuzzy Matching sehingga* diperlukan optimisasi atau langkah yang lebih kreatif untuk memberikan hasil analisis yang lebih komprehensif dan juga dengan pemrosesan yang efisien.

V. LAMPIRAN

Implementasi program yang digunakan pada makalah ini dapat dilihat pada link berikut: https://github.com/AgungLucker/Makalah-Stima-13523023

VI. UCAPAN TERIMA KASIH

Pertama-tama, penulis hendak ucapkan puji dan syukur kepada Tuhan Yang Maha Esa atas berkat, dan rahmat Nya sehingga makalah ini bisa terselesaikan. penulis juga mengucapkan terima kasih kepada Ibu Dr. Nur Ulfa Maulidevi, S.T, M.Sc. selaku dosen pengampu mata kuliah IF2211 Strategi Algoritma kelas K-1 beserta Bapak Dr. Ir. Rinaldi Munir, M.T., yang telah membimbing dan memberikan materi-materi ilmu yang berharga sepanjang penulisan makalah ini. Tak lupa, penulis juga sampaikan terima kasih kepada keluarga, teman-teman, dan pihak lain yang turut mendukung penulis selama proses penulisan makalah ini. Semoga Tuhan Yang Maha Esa membalas semua kebaikan yang diberikan dengan kebaikan yang berlipat ganda.

REFERENSI

[1] "Austronesian Languages," [Online]. Available: https://www.languagesgulper.com/eng/Austronesian.html [Diakses 23 Juni 2025].

[2] "Indonesian as an Austronesian Language," [Online]. Available: https://indonesian-online.com/indonesian-as-an-austronesian-language/ [Diakses 23 Juni 2025].

[3] "Regular Expression Language - Quick Reference, " [Online]. Available: https://learn.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference [Diakses 23 Juni 2025].

[4] M. L. Khodra , "String Matching dengan Regular Expression," 2025. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/24-String-Matching-dengan-Regex-(2025).pdf [Diakses 23 Juni 2025].

[5] Aerospike , "Introduction to fuzzy matching," .[Online]. Available: https://aerospike.com/blog/fuzzy-matching/ [Diakses 23 Juni 2025].

[6] R. Munir, "Program Dinamis (Dynamic Programming) - Bagian 1," 2025. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/25-Program-Dinamis-(2025)-Bagian1.pdf [Diakses 24 Juni 2025].

[7] E. Nam, "Understanding the Levenshtein Distance Equation for Beginners," 2019. [Online]. Available: https://medium.com/@ethannam/understanding-the-levenshtein-distance-equation-for-beginners-c4285a5604f0 [Diakses 24 Juni 2025]

[8] R. Munir, "Pencocokan String - (String/Pattern Matching)," 2025. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf [Diakses 24 Juni 2025].

[9] A. Pereltsvaig, "Do you speak Hawaiian?," 2019. [Online]. Available: https://www.languagesoftheworld.info/oceania-and-madagascar/speak-hawaiian.html [Diakses 24 Juni 2025]

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, 24 Juni 2025

Muhammad Aufa Farabi 13523023