Koreksi Ejaan Kata Bahasa Indonesia Menggunakan Konteks Kalimat dengan Program Dinamis dan Bigram

Aulia Adila - 13519100
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13519100@std.stei.itb.ac.id

***Abstract*—Bahasa sudah menjadi sebuah alat komunikasi paling efektif yang digunakan oleh manusia. Namun, penggunaan bahasa yang benar harus selalu diterapkan untuk menghindari kesalahpahaman terkait informasi yang disampaikan. Beberapa aspek yang berkaitan erat dengan kejelasan informasi adalah ketepatan ejaan kata serta kecocokan pemilihan kata dalam konteks kalimat. Ejaan kata yang dimaksud adalah ketepatan penulisan kata, yang kerap mengalami kesalahan tipografi atau *typographic error*. Dalam makalah ini, penulis melakukan identifikasi *typographic error* dan melakukan koreksi ejaan dengan memanfaatkan konsep *Levenshtein distance* (yang merupakan salah satu tipe *edit distance*) dalam program dinamis, sedangkan identifikasi kesesuaian pemilihan kata dalam konteks kalimat dilakukan dengan memanfaatkan kecocokan bigram.**

Kata kunci—program dinamis; Levenshtein distance; edit distance; bigram; ejaan kata; konteks kalimat

I. PENDAHULUAN

Tak dapat dipungkiri, bahasa sudah menjadi sebuah alat komunikasi paling efektif yang digunakan sejak zaman dahulu kala. Sejak penemuan pertama, perkembangan bahasa menjadi sangat cepat dan pesat. Kemampuan berbahasa manusia meningkat seiring dengan penyebaran dan pengimplementasiannya yang dilakukan secara oral/lisan maupun verbal/tulisan. Hal ini diperkuat dengan kemajuan teknologi yang semakin pesat, yang turut berperan sebagai media komunikasi efektif bagi manusia.

Dalam komunikasi, ketepatan dan kejelasan informasi menjadi sebuah hal yang penting untuk menciptakan komunikasi yang baik dan efektif. Penyajian informasi yang baik —dalam hal ini adalah ketepatan dan kejelasan informasi— ditunjang oleh ketepatan ejaan dari kata-kata yang digunakan serta kecocokan pemilihan kata dalam konteks kalimat. Ejaan kata dalam jenis komunikasi verbal berhubungan erat dengan penulisan kata, yaitu cara penyusunan serangkaian huruf sedemikian rupa hingga menghasilkan sebuah makna kata yang diinginkan. Sedangkan konteks kalimat berhubungan erat dengan penempatan kata dalam kalimat sedemikian rupa hingga menghasilkan makna yang padu.

Kesalahan ejaan kata –dalam hal ini dikhususkan pada kesalahan penulisan kata– kerap terjadi dalam kehidupan sehari-hari. Kesalahan ini seringkali disebut sebagai *typographical error* (yang disingkat sebagai *typo*), atau *misprint*. Kesalahan penulisan mayoritas tidak berdampak besar terhadap penyampaian informasi, namun sejarah pernah mencatat beberapa kasus besar yang terjadi akibat kesalahan jenis ini. Salah satu contoh kasus besar tersebut adalah ledakan roket NASA yang meledak pada tahun 1962 akibat kesalahan penempatan *hypen*.

Kesalahan penempatan kata dalam sebuah kalimat juga kerap terjadi pada sebuah proses komunikasi. Kesalahan ini berhubungan erat dengan kepaduan kalimat yang memiliki kesalahan konteks sehingga berpengaruh pada perbedaan penyampaian makna sebuah kalimat. Bagaimanapun, kesalahan penulisan dan kesalahan penempatan kata dalam sebuah kalimat sebaiknya dihindari oleh para pelaku komunikasi untuk mengurangi risiko kesalahan pemberian informasi yang dapat mengarah pada kesalahan pemahaman terhadap informasi.

Dalam makalah ini, penulis memanfaatkan metode program dinamis untuk melakukan koreksi terhadap kesalahan ejaan kata dalam bahasa Indonesia. Aplikasi program dinamis ini didasarkan pada konsep *edit distance* dengan menggunakan salah satu tipenya, yaitu *Levenshtein Distance*. Sedangkan untuk pengujian konteks kalimat dilakukan dengan pencocokan bigram antara masukan pengguna dengan korpus bahasa Indonesia.

II. DASAR TEORI

A. Program Dinamis

Program Dinamis (atau yang biasa disebut *dynamic programming*) meurpakan sebuah metode penyelesaian masalah dengan mengurai persoalan ke dalam beberapa subpersoalan yang tidak independen (setiap subpersoalan dapat menurunkan subpesoalan lain dan saling berkaitan). Algoritma program dinamis hanya menyelesaikan setiap subpersoalan sebanyak satu kali yang kemudian disimpan nilainya dalam sebuah matriks, untuk menghindari perhitungan ulang jawaban setiap kemunculan subpersoalan baru. Pencarian solusi menggunakan program dinamis dilakukan dengan menggunakan tabel yang dapat berkembang.

Pada umumnya, program dinamis diaplikasikan untuk persoalan optimasi, di mana akan dipilih solusi optimum dari sekian banyak calon solusi. Solusi optimum dapat berupa solusi

dengan nilai maksimum atau minimum. Pemecahan persoalan dengan algoritma program dinamis dapat dipecah ke dalam beberapa tahap sebagai berikut.

1. Identifikasi karakter dari solusi optimal.
2. Tentukan nilai dari solusi optimal secara rekursif.
3. Hitung nilai solusi optimal menggunakan pendekatan yang dipilih (bottom-up atau top-down).
4. Susun solusi optimal berdasarkan hasil komputasi.

Terdapat dua pendekatan yang dapat dilakukan untuk menyelesaikan persoalan dengan program dinamis. Pendekatan pertama adalah secara maju (*forward* atau *up-down*) yaitu perhitungan dilakukan dari tahap 1, 2, …, n-1, n. Sedangkan pendekatan kedua adalah secara mundur (*backward* atuai *bottom-up*) yaitu perhitungan dilakukan dari tahap n, n-1, …, 2, 1.

B. Edit distance

Edit distance merupakan studi kasus terbaik untuk pemecahan masalah pencocokan string yang mencari seberapa berbeda sebuah *string a* terhadap *b* dengan menghitung "jarak" antara *string*. Cara ini menentukan kuantifikasi perbedaan antara dua *string* dengan menghitung jumlah operasi minimum untuk transformasi *string* tersebut. Secara formal, *edit distance d(a,b)* adalah serangkaian *edit operation* dengan bobot minimum untuk mentransformasikan *a* menuju *b*. Perbedaan definisi dari *edit distance* mempengaruhi jenis operasi yang digunakan.

Operations adalah sejumlah hingga aturan dengan bentuk $\delta(z, w) = t$, di mana *z* dan *w* adalah *string* yang berbeda dan *t* adalah bilangan riil nonnegatif. Ketika operasi tersebut berhasil mengubah *z* menjadi *w*, maka tidak terdapat operasi lain yang dapat dilakukan terhadap *w*. Beberapa jenis operasi yang dapat diaplikasikan adalah sebagai berikut.

a. *Insertion*: $\delta(\varepsilon, a)$, menyisipkan huruf *a*
b. *Deletion*: $\delta(a, \varepsilon)$, menghapus huruf *a*
c. *Substitution* atau *replacement*: $\delta(a, b)$ dengan $a \neq b$, mengganti *a* dengan *b*
d. *Transposition*: $\delta(ab, ba)$ dengan $a \neq b$, menukar huruf *a* dan *b* yang berdekatan

Terdapat beberapa tipe *edit distance* yang masing-masing memiliki *string operation* yang berbeda. Penjabarannya adalah sebagai berikut.

a. *Levenshtein distance* (dalam beberapa referensi, istilah *Levenshtein distance* disamakan dengan *edit distance*) menggunakan *deletion*, *insertion*, dan *substitution*.
b. *Longest common subsequence* menggunakan *insertion* dan *deletion*, namun bukan *substitution*.
c. *Hamming distance* hanya menggunakan *substitution*, sehingga hanya berlaku untuk *string* dengan panjang yang sama.
d. *Damerau-Levenshtein distance* menggunakan *insertion*, *deletion*, *substitution*, dan *transposition* dari dua huruf berdekatan.
e. *Jaro distance* hanya menggunakan *transposition*.

Dalam bidang ilmu komputer, algoritma program dinamis yang digunakan untuk menghitung *edit distance* dari dua string berbeda adalah algoritma *Wagner-Fischer*. Hasil dari perhitungan adalah berupa *edit distance* dengan tipe *Levenshtein distance*. Maka dari itu, algoritma *Wagner-Fischer* juga kerap disebut algoritma *Levenshtein*.

C. Levenshtein Distance

Levenshtein Distance adalah sebuah metrik *string* yang digunakan untuk mengukur perbedaan dari dua sekuens, yang umumnya mengukur perbedaan dua *string* atau sekuens karakter. *Levenshtein distance* mengukur jumlah minimum dari *edit operations* yang dilakukan untuk mengubah sebuah *string* menuju *string* lainnya. Konsep ini diperkenalkan oleh Vladimir Levenshtein pada tahun 1965.

Terdapat tiga jenis *edit operation* yang dapat dilakukan dalam penerapan konsep *Levenshtein Distance*, yaitu sebagai berikut.

1. *Deletion*, yaitu operasi penghapusan karakter.
2. *Insertion*, yaitu operasi penyisipan karakter.
3. *Substitution*, yaitu operasi penggantian karakter.

Berikut adalah ilustrasi ketiga edit operasi dalam algoritma *Levenshtein* yang diimplementasikan dalam pengubahan *string* FRESH menjadi FRIED.

F	R	E	S	H
F	R	I	E	D

Gambar 1. Perbandingan dua *string (Sumber: milik penulis)*

1. Karakter F dan R dalam kedua *string* sudah sama, sehingga tidak perlu dilakukan *edit*.
2. Dilakukan *insertion* karakter I, sehingga *string* FRESH mengalami perubahan sebagai berikut.

F	R	I	E	S	H

Gambar 2. Contoh hasil *edit operation insertion* pada *string (Sumber: milik penulis)*

3. Karakter E dalam kedua *string* sudah sama, sehingga tidak perlu dilakukan *edit*.
4. Dilakukan *substitution* karakter D, sehingga *string* FRIESH mengalami perubahan sebagai berikut.

F	R	I	E	D	H

Gambar 3. Contoh hasil *edit operation subtitution* pada *string (Sumber: milik penulis)*

5. Dilakukan *deletion* karakter H, sehingga *string* FRIEDH mengalami perubahan sebagai berikut.

F	R	I	E	D

Gambar 4. Contoh hasil *edit operation deletion* pada *string (Sumber: milik penulis)*

Sehingga dapat disimpulkan bahwa *Levenshtein distance* bernilai 3.

Misalkan terdapat *string a* dan *b* dengan panjang *string* dinotasikan dalam |a| dan |b|, dapat dihitung *Levenshtein distance* menggunakan fungsi $lev_{a,b}(i,j)$ dengan *i* adalah indeks terminal penunjuk karakter dalam *string a* dan *j* adalah indeks terminal penunjuk karakter dalam *string b*.

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

Gambar 5. Fungsi Levenshtein Distance
(*Sumber*:https://dzone.com/articles/the-levenshtein-algorithm-1)

Fungsi ini diimplementasikan dalam sebuah algoritma Wagner-Fischer, atau dapat disebut algoritma Levenshtein. Algoritma ini memanfaatkan metode program dinamis dengan pendekatan *top down*. Komputasi dilakukan dengan membuat matriks yang ukurannya sesuai dengan panjang string untuk menyimpan Levenshtein distance antara seluruh prefix string a dan seluruh prefix string b. Berikut adalah pseudocode fungsi perhitungan Levenshtein distance.

```
function levDist (str_a: string, str_b: string) → integer
   input : string a,
           string b
   output: Levenshtein distance of string a
           and string b

//Make matrix
ly ← length(str_a) + 1
lx ← length(str_b) + 1
declare int matrix[ly,lx]
set each element in matrix to zero

//Fill in row and column with basis
//if minimum(x,y) is 1 then matrix[x,y]<-maximum(x,y)
for i from 1 to length(str_a):
   matrix[i,0] ← i
for j from 1 to length(str_b):
   matrix[0,j] ← j

//Fill in cells in matrix using dynamic programming
for i from 1 to length(str_a):
   for j from 1 to length(str_b):
        if str_a[i] ≠ str_b[j] then
                add ← 1
        else
                add ← 0
//Implement Levenshtein distance function
        matrix[i,j] ← minimum(matrix[i-1,j] + 1,
                              matrix[i,j-1] + 1,
                              matrix[i-1,j-1] + add)
//return Levenshtein distance of str_a and str_b
return matrix[length(str_a), length(str_b)]
```

Gambar 6. Pseudocode Algoritma *Levenshtein (Sumber: milik penulis)*

Sebagai ilustrasi, akan dihitung Levenshtein distance dari 2 buah string, yaitu FRESH dan FRIED. Langkah awal yang dilakukan adalah pembuatan matriks sebagai berikut.

		F	R	I	E	D
	0	1	2	3	4	5
F	1					
R	2					
E	3					
S	4					
H	5					

Gambar 7. Matriks kosong 5x5 *(Sumber: milik penulis)*

Pengisian sel pada matriks dimulai dari indeks (1,1). Karena nilai minimum(1,1) bukan sama dengan nol, maka akan digunakan fungsi Levenshtein distance. Perhitungan adalah sebagai berikut.

$$\begin{aligned} lev_{a,b}(i-1,j)+1 &= lev_{a,b}(0,1)+1 \\ &= 1+1=2 \\ lev_{a,b}(i,j-1)+1 &= lev_{a,b}(1,0)+1 \\ &= 1+1=2 \\ lev_{a,b}(i-1,j-1)+1 &= lev_{a,b}(0,0)+0 \\ &= 0 \end{aligned}$$

Nilai minimum dari ketiga Levenshtein distance di atas adalah 0. Maka dari itu, sel pada (1,1) diisi 0. Perhitungan ini dilakukan hingga seluruh sel dalam matriks penuh terisi seperti berikut.

		F	R	I	E	D
	0	1	2	3	4	5
F	1	0	1	2	3	4
R	2	1	0	1	2	3
E	3	2	1	1	1	2
S	4	3	2	2	2	2
H	5	4	3	3	3	3

Gambar 7. Matriks terisi 5x5 *(Sumber: milik penulis)*

Nilai yang diambil sebagai hasil akhir perhitungan Levenshtein distance dari dua string adalah nilai pada sel paling kanan bawah. Dalam persoalan ini, nilai tersebut adalah 3. Sehingga, dapat disimpulkan Levenshtein distance dari kata FRESH dan FRIED adalah 3.

D. Bigram

Bigram adalah sebuah sekuens dari dua elemen bersisian dari sebuah *token*, yang umumnya berupa huruf, suku kata, atau kata. Bigram merupakan salah satu bentuk dari N-gram dengan nilai n adalah 2. Misalkan sebuah kata 'the' memiliki bigram ('t','h') dan ('h','e'), ataupun sebuah kalimat 'halo halo bandung' memiliki bigram ('halo','halo') dan ('halo', 'bandung').

Bigram sering diaplikasikan dalam model bahasa untuk pengenalan suara, sebagai sebuah kasus khusus dari N-gram. Frekuensi bigram terhadap korpus juga dapat ditentukan, dan nilai ini dapat digunakan dalam bidang kriptografi untuk menyelesaikan persoalan kriptografi. Selain itu, frekuensi bigram juga dapat digunakan dalam indentifikasi bahasa statistik.

E. Ejaan Kata dan Konteks Kalimat

Kata merupakan satuan bahasa yang memiliki arti atau sebuah pengertian, dan terdiri atas satu atau lebih morfem. Umumnya, sebuah kata dapat memiliki akar data dengan beberapa atau tanpa afiks. Penyampaian sebuah kata dalam bentuk oral maupun verbal sangat berkaitan erat dengan ejaan. Menurut Kamus Besar Bahasa Indonesia (KBBI), ejaan adalah kaidah cara menggambarkan bunyi-bunyi(kata, kalimat, dan sebagainya) dalam tulisan (huruf-huruf) serta penggunaan tanda baca (KBBI, 2008:353). Ejaan terkait dengan tata tulis yang meliputi pemakaian huruf, penulisan kata, dan pemakaian tanda baca.

Kalimat adalah satuan bahasa berupa kata atau rangkaian kata yang dapat berdiri sendiri. Sebuah kalimat, baik kalimat tunggal maupun majemuk, dapat terdiri atas beberapa frasa maupun klausa yang saling tersusun membentuk sebuah kalimat padu dengan makna yang utuh dan lengkap. Bagian-bagian dari kalimat tersebut –yang dapat diwakili oleh susunan dua kata berurut atau lebih– memiliki konteks, yang dapat mendukung atau menambahkan kejelasan makna. Untuk membentuk sebuah kalimat dengan makna yang tepat, dibutuhkan susunan dan pemilihan kata yang tepat. Susunan kata yang tidak sesuai dengan konteks kalimat dapat mengubah makna kalimat secara keseluruhan.

Ejaan kata dan konteks kalimat merupakan dua hal yang wajib diperhatikan dalam penyampaian informasi dalam proses komunikasi, untuk menghindari kesalahan pemaknaan dan pengertian terhadap informasi.

III. PEMBAHASAN

Penyelesaian masalah kesalahan ejaan kata bahasa Indonesia menggunakan konteks kalimat memanfaatkan beberapa konsep dan algoritma yang akan diuraikan dalam bab ini. Terdapat beberapa batasan untuk dapat memenuhi tujuan akhir program, yaitu menghasilkan kalimat ataupun kata yang merupakan hasil koreksi ejaan dari masukan user. Beberapa batasan tersebut di antaranya adalah sebagai berikut.

1. Masukan pengguna tidak memiliki tanda baca titik (.)
2. Ejaan kata selalu benar pada karakter pertama
3. Bahasa yang digunakan harus merupakan bahasa Indonesia

Berikut adalah beberapa langkah utama pemecahan masalah untuk koreksi ejaan kata bahasa Indonesia menggunakan konteks kalimat.

A. Pembuatan Daftar Kata dan Bigram Bahasa Indonesia

Sebelum diolah dalam beberapa rangkaian algoritma, perlu disiapkan beberapa struktur data yang akan menjadi data acuan dalam proses komputasi. Sumber data diambil dari https://github.com/kirralabs/indonesian-NLP-resources, yaitu dataset kalimat *leipzig indonesian sentence collection*. Dataset kata dan kalimat dalam bentuk txt kemudian diolah menjadi dua struktur data utama. Struktur pertama adalah daftar kata bahasa Indonesia dalam bentuk *dictionary*. Sedangkan struktur kedua adalah daftar bigram dalam bentuk *dictionary* yang diolah dari korpus kalimat bahasa Indonesia. Dalam mengolah korpus data, penulis memanfaatkan beberapa fungsional dalam library nltk. Algoritma pengolahan data adalah sebagai berikut.

```
def makedictKata(filename1):
    fkata = open(filename1, 'r', encoding='utf-8')
    dict_kata = {}
    for line in fkata:
        w = line.split("\t")
        l = w[1:3]
        dict_kata[l[0]] = l[1].rstrip('\n')
    return dict_kata
```

Gambar 8. Implementasi fungsi penyusunan struktur data daftar kata dalam bahasa Indonesia dalam bahasa Python *(Sumber: milik penulis)*

```
def bigramCorpus(filename2):
    arr_kalimat = []
    fkalimat = open(filename2, 'r', encoding='utf-8')
    for line in fkalimat:
        w = line.split("\t")
        arr_kalimat.append(w[1])

    dict_bigramCorpus = {}
    for line in arr_kalimat:
        token = nltk.word_tokenize(line)
        line_bigram = list(ngrams(token,2))
        for bigram in line_bigram:
            if (bigram not in dict_bigramCorpus):
                dict_bigramCorpus[bigram] = 1
            else :
                dict_bigramCorpus[bigram] += 1

    return dict_bigramCorpus
```

Gambar 9. Implementasi fungsi penyusunan struktur data daftar bigram dalam bahasa Indonesia dalam bahasa Python *(Sumber: milik penulis)*

B. *Implementasi Algoritma Levenshtein*

Program menerima masukan dari pengguna berupa rangkaian kata yang membentuk kalimat. Masukan tersebut diproses dengan memanfaatkan library ntlk, kemudian diidentifikasi kesesuaian ejaannya dengan cara melakukan iterasi pemeriksaan terhadap daftar kata bahasa Indonesia (dalam bentuk *dictionary*) yang telah disusun sebelumnya. Kata yang teridentifikasi memiliki kesalahan ejaan (disebut sebagai kata uji) akan diproses dalam sebuah algoritma sederhana untuk menghasilkan daftar kata yang memiliki kesamaan karakter pertama dengan kata uji.

Daftar kata diproses ke dalam algoritma *Levenshtein*, untuk mencari daftar kata yang memiliki nilai *Levenshtein distance* minimum. *Algoritma Levenshtein* diimplementasikan dalam bahasa Python sesuai dengan pseudocode (lihat Gambar …). Dalam implementasi, dimanfaatkan library Numpy dari Python untuk mempermudah proses perhitungan dalam representasi matriks yang disusun.

```
import numpy as np

def levDist(str_a, str_b):
    #Membuat matriks dengan ukuran (panjang string + 1)
    ly = len(str_a) + 1
    lx = len(str_b) + 1
    matrix = np.zeros((ly,lx))

    #Implementasi basis, yaitu mengisi nilai
    #matriks[x,y] dengan max(x,y) jika min(x,y) adalah 1
    for i in range (ly):
        matrix[i,0] = i
    for j in range (lx):
        matrix[0,j] = j

    #Mengisi matriks menggunakan program dinamis
    for i in range (1,ly):
        for j in range (1, lx):
            if (str_a[i-1] != str_b[j-1]):
                 add = 1
            else:
                add = 0
            #Implementasi fungsi Levenshtein distance
            matrix[i,j] = min  (matrix[i-1,j] + 1,
                                matrix[i,j-1] + 1,
                                matrix[i-1, j-1] + add)
            print(int(matrix[i,j]),end=" ")
        print("\n")
    #Mengembalikan nilai Levenshtein distance
    return (int(matrix[ly-1,lx-1]))
```

Gambar 10. Implementasi algoritma *Levenshtein* dalam bahasa Python *(Sumber: milik penulis)*

Algoritma ini diuji untuk beberapa kata berbeda. Hasil pengujian ditunjukkan dalam tabel berikut.

Tabel 1. Hasil pengujian algoritma *Levenshtein* terhadap beberapa kata

Kata pertama	Kata kedua	Hasil Pengujian
fresh	fried	Masukkan kata pertama: fresh Masukkan kata pertama: fried Nilai Levenshtein distance: 3
levenshtein	meilenstein	Masukkan kata pertama: levenshtein Masukkan kata pertama: meilenstein Nilai Levenshtein distance: 4

C. *Pemeriksaan Kesesuaian Konteks Kalimat Menggunakan Bigram*

Terdapat kasus khusus di mana kata dengan nilai *Levenshtein distance* yang sama memiliki jumlah kata lebih dari satu. Untuk memilih kata yang paling sesuai, dipertimbangkan aspek kesesuaian kata terhadap konteks kalimat. Konteks kalimat dalam hal ini diwakili oleh bentuk bigram dari kata sebelum dan kata sesudah kandidat kata. Beberapa uraian langkah untuk memeriksa kesesuaian kata terhadap konteks kalimat dengan memanfaatkan bigram adalah sebagai berikut.

1. Iterasi seluruh kandidat kata (daftar kata yang memiliki nilai *Levenshtein distance* minimum yang sama). Lakukan pemrosesan untuk setiap kandidat kata.
2. Bentuk dua bigram dari kandidat kata terkait, serta kata sebelum dan sesudah kandidat kata (yang diambil dari kalimat pengguna). Misal terdapat kandidat kata ‘aku’ dan ‘akun’ yang berasal dari kalimat ‘Kemarin akuw pergi’. ‘akuw’ diidentifikasi sebagai kata dengan ejaan yang salah, kemudian hasil analisis kata bahasa Indonesia dengan Levenshtein distance minimum adalah ‘aku’ dan ‘akun’.

   Bigram dari kandidat kata ‘aku’ : (‘Kemarin’, ‘aku’) dan (‘aku’, ‘pergi’)

   Bigram dari kandidat kata ‘akun’ : (‘Kemarin’, ‘akun’) dan (‘akun’, ‘pergi’)
3. Dilakukan pemeriksaan kesesuaian masing-masing bigram yang telah terbentuk terhadap daftar bigram bahasa Indonesia yang telah disusun sebelumnya. Bigram yang terpilih adalah bigram yang memiliki frekuensi kemunculan tertinggi dalam daftar bigram.

D. *Pemilihan Kata yang Merupakan Hasil Koreksi Ejaan*

Berdasarkan algoritma *Levenshtein* dan pengujian kesesuaian bigram, dihasilkan kata yang merupakan hasil koreksi ejaan dari kata uji. Kata tersebut merupakan kata dengan susunan karakter terdekat dengan kata uji, yang dibuktikan dengan nilai Levenshtein distance kata terpilih yang telah diuji

minimalitasnya. Jika terdapat lebih dari satu kata dengan nilai Levensthtein distance yang sama, maka kesesuaian kata terhadap konteks kalimat dijadikan salah satu pertimbangan. Konsep tersebut diimplementasi dengan pemanfaatan konsep bigram. Kata terpilih merupakan kata yang terkandung dalam bigram dengan nilai frekuensi kemunculan tertinggi pada daftar bigram bahasa Indonesia.

Dalam penyajian jawaban, penulis menampilkan kembali kalimat masukan pengguna yang sudah dikoreksi ejaannya.

```
def getAnswer(LevDist, dict_bigramCorpus, userToken, i):
    # Mencari nilai Levenshtein distance terkecil
    minLev = min(LevDist.items(), key=lambda item: item[1])[1]
    print("Jarak terpendek : ", minLev)

    # Array yang menyimpan daftar kandidat jawaban
    # Kandidat jawaban adalah daftar kata dengan
    # nilai Levenshtein distance yang sama
    toBeTested = []

    for k,v in LevDist.items():
        if (v == minLev):
            toBeTested.append(k[1])

        if (len(toBeTested) > 1):
            maxBigram = -1
            ans = 'test'
            # Iterasi terhadap seluruh kandidat jawaban
            for test in toBeTested:
                countBigram = 0
                # Membuat bigram dari setiap kandidat jawaban
                bigram_before = wordBefore(userToken,test,i)
                bigram_after = wordAfter(userToken,test,i)

                if (bigram_before in dict_bigramCorpus):
                   countBigram +=
                              dict_bigramCorpus[bigram_before]
                if (bigram_after in dict_bigramCorpus):
                   countBigram +=
                               dict_bigramCorpus[bigram_after]

                # Kata terpilih adalah kata yg terkandung dlm
                # bigram dengan frekuensi kemunculan terbanyak
                # (dalam daftar bigram bahasa Indonesia)
                if (countBigram > maxBigram):
                    maxBigram = countBigram
                    ans = test #jawaban
        else:
            ans = toBeTested

    return ans
```

Gambar 11. Implementasi pemeriksaan kesesuaian kata dengan konteks kalimat serta pemilihan kata sebagai jawaban akhir dalam bahasa Python *(Sumber: milik penulis)*

IV. Penutup

A. Kesimpulan

Konsep program dinamis dapat diaplikasikan dalam berbagai disiplin ilmu. Konsep ini bersifat fleksibel, karena berkaitan erat dengan efisiensi penyelesaian persoalan yang menggunakan optimasi. Terdapat pula konsep edit distance, yaitu sebuah studi kasus terbaik dalam hal analisis perbedaan antara dua buah string. Pemilihan kata sebagai hasil koreksi dari ejaan kata dapat dilakukan dengan memanfaatkan algoritma Levenshtein yang menghasilkan nilai Levenshtein distance (dapat pula disebut edit distance) sebagai salah satu algoritma untuk mengidentifikasi perbedaan string yang memanfaatkan konsep program dinamis. Untuk menambah akurasi pemilihan kata, dimanfaatkan konsep bigram yang merepresentasikan konteks kalimat. Dengan demikian, masalah kesalahan ejaan kata dalam kalimat bahasa Indonesia dapat diselesaikan dengan konsep Levenshtein distance yang dipadukan dengan konsep bigram.

B. Saran

Identifikasi perbedaan antara string dapat dilakukan dengan pendekatan lain (selain program dinamis), atau pendekatan yang menggabungkan konsep dan algoritma lainnya. Beberapa pendekatan yang dapat diuji adalah Divide and Conquer, String Matching (algoritma Knuth-Morris-Pratt atau algoritma Boyer-Moore), Regular Expression, dan sebagainya.

Daftar kata dan bigram juga dapat perkaya dengan menggunakan dataset yang lebih banyak dan beragam. Pengambilan data juga dapat dilakukan menggunakan DBMS untuk ukuran data yang besar. Selain itu, pengembangan terhadap penyajian informasi dapat memanfaatkan *Graphical User Interface* atau *User Interface* berbasis *Web*.

Link Video Youtube

Penulis membuat video mengenai makalah ini yang dapat diakses pada tautan berikut.

https://youtu.be/asDFt_rZmCQ

Ucapan Terima Kasih

Penulis ingin menghaturkan puji dan syukur kepada Tuhan Yang Maha Esa sebab atas rahmat dan karunia-Nya, makalah ini dapat diselesaikan dengan baik. Penulis juga ingin mengucapkan terima kasih sebanyak-banyaknya kepada Ibu Dr. Nur Ulfa Maulidevi, ST., M.Sc. sebagai dosen pengampu K2, Bapak Dr. Ir. Rinaldi, MT., Bapak Dr. Ir. Rila Mandala, M. Eng, serta Bapak Prof. Ir. Dwi Hendratmo Widyantoro, M.Sc., Ph.D. atas bimbingannya selama satu semester perkuliahan ini. Semoga seluruh penyampaian ilmu dan pengorbanan dapat menjadi amal jariyah bagi bapak dan ibu. Tak lupa penulis juga ingin mengucapkan terima kasih kepada seluruh pihak lain yang terlibat dalam penyelesaian makalah ini.

REFERENSI

[1] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C. (2001), Introduction to Algorithms (2nd ed.), MIT Press & McGraw–Hill, ISBN 0-262-03293-7 . pp. 344.

[2] Navarro, Gonzalo (2001). "A guided tour to approximate string matching" (PDF). ACM Computing Surveys. 33 (1): 31–88. CiteSeerX 10.1.1.452.6317. doi:10.1145/375360.375365. S2CID 207551224

[3] Corbin, Kyle (1989) "Double, Triple, and Quadruple Bigrams," Word Ways: Vol. 22 : Iss. 3 , Article 8.

[4] IF2211 Strategi Algoritma (2020). Program Dinamis (Dynamic Programming). KK Informatika STEI ITB. (diakses melalui https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Program-Dinamis-2020-Bagian1.pdf pada 10 Mei 2021 pukul 11:22 WIB)

[5] Bigram (2021) (Diakses melalui https://en.wikipedia.org/wiki/Bigram pada 10 Mei 2021 pukul 21.30 WIB)

[6] The Levenshtein Algorithm (2017) (Diakses melalui https://www.cuelogic.com/blog/the-levenshtein-algorithm pada 10 Mei 2021 pukul 17.55 WIB)

[7] Levenshtein Distance (2019) (Diakses melalui https://devopedia.org/levenshtein-distance#sample-code pada 10 Mei 2021 pukul 18.45 WIB)

[8] Pusat Pembinaan Badan Pengembangan dan Pembinaan Bahasa Kementerian Pendidikan dan Kebudayaan (2015). “Seri Penyuluhan Bahasa Indonesia EJAAN”(PDF). Halaman 6-7.

[9] Kamus Besar Bahasa Indonesia (2021) (Diakses melalui https://kbbi.web.id/ pada 10 Mei 2021 pukul 23.14 WIB)

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, 11 Mei 2021

Aulia Adila
13519100