BAB II
KAJIAN PUSTAKA DAN DASAR TEORI

Bab ini berisi kajian pustaka dan dasar teori yang berhubungan dengan penggunaan *string matching* dalam pengembangan Sistem Penilaian Esai Otomatis. Kajian pustaka membahas penelitian yang telah ada dan dasar teori membahas teori yang diperlukan untuk menyusun penelitian.

2.1. Kajian Pustaka

Terdapat beberapa metode dalam penilaian esai otomatis (*essay grading*) yang saat ini telah dikembangkan baik untuk kebutuhan riset ataupun komersial. Metode-metode tersebut menunjukkan tingkat korelasi yang cukup tinggi bila dibandingkan dengan pemeriksaan manual. Setiap metode memiliki teknik penilaian yang berbeda walaupun teknik penilaian yang dilakukan berbeda-beda tujuan yang dicapai sama, yaitu membangun sistem yang mampu memberikan penilaian terhadap jawaban esai secara otomatis seobjektif mungkin [HER-08].

Dalam konsep ujian esai, pelaksanaan ujian dapat dilakukan, mulai dari menjawab soal ujian hingga proses penilaian. Selama ini kebanyakan proses ujian esai dan penilaiannya dilaksanakan secara manual yaitu dengan membaca esai satu per satu. Para pengajar perlu menghabiskan banyak waktu untuk menilai jawaban ujian siswa mereka. Semakin banyak jumlah ujian yang dikoreksi, kualitas penilaian yang diberikan semakin menurun [AND-08].

Pengembangan sistem penilaian jawaban esai telah dilakukan semenjak tahun 2003 oleh Valenti, Neri, dan Cucchiarelli (2003). Sistem yang dikembangkan bernama Conceptual rater (C-Rater). Sistem ini menghasilkan analisis dari relasi logika antara komponen sintaksis yang muncul pada setiap kalimat jawaban esai yang menangkap konsep yang disampaikan. Sistem ini hanya menggunakan satu kunci jawaban sebagai pembanding setiap jawaban peserta ujian.

Penilaian jawaban esai di Indonesia sendiri pengembangan sistem dimulai sejak tahun 2005 yang dilakukan oleh Krisnanda (2005). Dalam penelitiannya, Krisnanda menggunakan metode Latent Semantic Analisys (LSA) dengan variasi

banyaknya kata kunci jawaban. Penelitian berikutnya dilakukan oleh Ratna, Budiarjo, dan Hartanti (2007). Sistem yang dikembangkan menerapkan implementasi LSA dan normalisasi Frobenius. Sistem ini disebut "Sistem Penilaian Esai Otomatis". Tingkat kesamaan hasil penilaian sistem dengan hasil penilaian manusia mencapai 69.80% sampai dengan 94.64% [HER-08].

Penelitian ini dikembangkan lagi dalam skripsi yang sudah banyak diterapkan beberapa diantaranya adalah: Rohmawati (2010) mengembangkan sistem penilaian esai otomatis pada *e-learning* dengan metode *cosine similarity*. Penerapan penilaian esai otomatis dilakukan dengan basis bahasa Inggris. Bahasa basis mempengaruhi proses dan hasil karena karakteristik dari bahasa yang berbeda. Sistem penilaian otomatis jawaban essay menggunakan ontologi pada *moodle* oleh Andi (2012), dalam penelitiannya menggunakan pengukuran similaritas *semantic* berbasis *Wordnet* pada pencarian sinonim. Hendri (2012) melakukan penelitian untuk mendeteksi kemiripan dokumen yang diperiksa menggunakan algoritma *Levenshtein Distance* dengan melakukan pengecekan semua kalimat pada dokumen satu dengan semua kalimat pada dokumen pembanding sehingga bisa ketemu perbandingan kalimat yang mempunyai kesamaan yang tinggi.

Kemudian David (2012) membandingkan kemiripan jawaban essai dengan kunci jawaban yang sebelumnya telah disimpan dalam sistem. Algoritma yang digunakan adalah algoritma *Rabin-Karp*, untuk pencarian lebih dari satu kata (*multi pattern*). Dengan mengetahui prosentase kemiripan kedua teks jawaban tersebut dapat dijadikan acuan untuk melakukan penilaian.

Dari uraian diatas, maka pada penelitian ini nantinya akan dicoba untuk menerapkan sistem penilaian jawaban esai otomatis menggunakan algoritma *Leveinshtein Distance.*

2.2. Ujian Esai

Ujian merupakan alat atau prosedur yang digunakan untuk mengetahui atau mengukur sesuatu dalam suasana, dengan cara dan aturan-aturan yang sudah ditentukan. Ujian diadakan untuk mengukur kemampuan belajar dari mahasiswa. Terdapat bermacam-macam ujian yang diadakan. Ujian esai merupakan tes yang

disusun dalam bentuk pertanyaan terstruktur dan mahasiswa menyusun, mengorganisasikan sendiri jawaban tiap pertanyaan itu dengan bahasa sendiri. Soal esai merupakan bentuk evaluasi dimana pilihan jawaban tidak disediakan, dan mahasiswa harus menjawab dengan kalimat, sehingga jawaban dapat sangat bervariasi sesuai dengan pemikiran masing-masing peserta ujian. Penilaian dengan esai (*essay grading*) tetap menjadi pilihan pengajar dalam mengevaluasi tingkat kemampuan dari siswanya walaupun kenyataannya tidak mudah untuk memberikan penilaian yang objektif pada jawaban setiap siswa. Bentuk ini oleh banyak peneliti dianggap alat yang sangat ampuh untuk menilai pencapaian hasil pembelajaran, begitu juga untuk mengamati kemahiran berpikir tingkat tinggi siswa, seperti sintesa dan analisa [ANA-06].

Menurut Susanti (2008) soal uraian (esai) berbeda dengan soal objektif dalam kebenarannya bertingkat. Jawaban tidak dinilai mulai dari 100% benar dan 100% salah. Kebenaran bertingkat tergantung tingkat kesesuaian jawaban siswa dengan jawaban yang dikehendaki yang dituangkan dalam kunci. Jawaban mungkin mengarah kepada jawaban yang tidak tunggal (*divergence*). Kebenaran yang dicapai bisa 0%, 20%, 50%, 70%, atau 100% tergantung dari ketepatan jawabannya [LES-12].

2.3. Text Mining

2.3.1. Pengertian *Text Mining*

Text mining memiliki definisi menambang data yang berupa teks dimana sumber data biasanya didapatkan dari dokumen, dan tujuannya adalah mencari kata-kata yang dapat mewakili isi dari dokumen sehingga dapat dilakukan analisa keterhubungan antar dokumen.

Text mining secara luas didefinisikan sebagai proses mencari tahu secara intensif dimana pengguna berinteraksi dengan kumpulan dokumen sepanjang waktu dengan menggunakan serangkaian alat analisis. Pada cara yang sejalan dengan data mining, *text mining* berusaha mengutip informasi yang berguna dari sumber data melalui identifikasi dan eksplorasi pola yang menarik. Akan tetapi pada *text mining*, sumber data adalah kumpulan dokumen dan pola yang menarik

tidak ditemukan pada record database yang terbentuk melainkan pada data kata per kata yang tidak terstruktur pada kumpulan dokumen tersebut. [FEL-07]

Tujuan utama teks *mining* adalah mendukung proses *knowledge discovery* pada koleksi dokumen yang besar. Pada prinsipnya, teks mining adalah bidang ilmu multidisipliner, melibatkan *information retrieval (IR), text analysis, information extraction (IE), clustering, categorization, visualization, database technology, natural language processing (NLP), machine learning,* dan *data mining*. Dapat pula dikatakan bahwa teks mining merupakan salah satu bentuk aplikasi kecerdasan buatan (*artificial intelligence / AI*).

Text mining bisa dianggap subjek riset yang tergolong baru. *Text mining* dapat memberikan solusi dari permasalahan seperti pemrosesan, pengorganisasian atau pengelompokkan dan menganalisa *unstructured text* dalam jumlah besar. Dalam memberikan solusi, *text mining* mengadopsi dan mengembangkan banyak teknik dari bidang lain, seperti *Data mining, Information Retrieval*, Statistik dan Matematik, *Machine Learning, Linguistic*, *Natural Languange Processing*, dan *Visualization*. Kegiatan riset untuk *text mining* antara lain ekstraksi dan penyimpanan teks, *preprocessing* akan konten teks, pengumpulan data statistik dan indexing dan analisa konten.

Perbedaan mendasar antara teks *mining* dan *data mining* terletak pada sumber data yang digunakan. Pada *data mining*, pola-pola diekstrak dari basis data yang terstruktur, sedangkan di teks *mining*, pola-pola diekstrak dari data tekstual (*natural language*). Secara umum, basis data didesain untuk program dengan tujuan melakukan pemrosesan secara otomatis, sedangkan teks ditulis untuk dibaca langsung oleh manusia.

2.3.2. Information Retrieval

Information Retrieval adalah "bidang di persimpangan ilmu informasi dan ilmu komputer. Berkutat dengan pengindeksan dan pengambilan informasi dari sumber informasi heterogen dan sebagian besar tekstual. Temu kembali informasi *(information retrieval*) adalah ilmu pencarian informasi pada dokumen, pencarian untuk dokumen itu sendiri, pencarian untuk metadata yang menjelaskan dokumen, atau mencari di dalam database, baik relasi database yang *stand-alone* atau

hipertext database yang terdapat pada *network* seperti internet atau *World Wide Web* atau *intranet*, untuk teks, suara, gambar, atau data. *Information retrieval (IR)* adalah ilmu yang lahir dari berbagai disiplin ilmu, baik ilmu komputer, matematika, ilmu kepustakaan, ilmu informasi, psikologi kognitif, linguistik, statistik, maupun fisika.

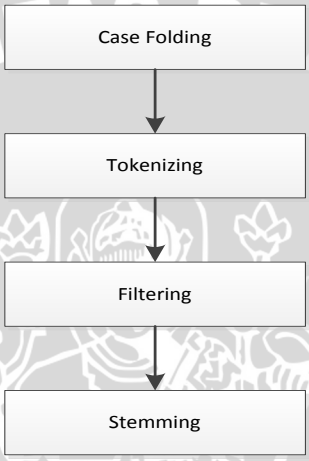
Secara prinsip, penyimpanan informasi dan penemuan kembali informasi adalah hal yang sederhana. Misalkan terdapat tempat penyimpanan dokumen-dokumen dan seseorang (user) merumuskan suatu pertanyaan (*request* atau *query*) yang jawabannya adalah himpunan dokumen yang mengandung informasi yang diperlukan yang diekspresikan melalui pertanyaan user. User bisa saja memperoleh dokumen-dokumen yang diperlukannya dengan membaca semua dokumen dalam tempat penyimpanan, menyimpan dokumen-dokumen yang relevan dan membuang dokumen lainnya. Hal ini merupakan *perfect retrieval*, tetapi solusi ini tidak praktis. Karena user tidak memiliki waktu atau tidak ingin menghabiskan waktunya untuk membaca seluruh koleksi dokumen, terlepas dari kenyataan bahwa secara fisik user tidak mungkin dapat melakukannya.

Information retrieval berbeda dengan *database retrieval*. Sistem *database retrieval* umumnya memberikan tepat semua dokumen atau objek yang memenuhi kriteria tertentu. Sedangkan sistem *information retrieval* melakukan suatu pendugaan atas dokumen-dokumen yang diinginkan pengguna, dengan melihat tingkat kemiripannya. Sistem *database retrieval* dapat menerima *query* yang kompleks dan memberikan semua jawaban sesuai kondisi logis dari *query* bersangkutan. Sedangkan sistem IR biasanya memberikan beberapa dokumen yang telah diurutkan berdasarkan tingkat kemiripannya dengan *query* yang diinputkan.

Information Retrieval System (IRS) tidak memberi tahu pengguna masalah yang ditanyakannya. Sistem tersebut hanya memberitahukan keberadaan dan keterangan dokumen yang berhubungan dengan permintaan pengguna. Dengan memakai bahasa natural sebagai bahasa *query*, IRS memberikan kemudahan kepada pengguna dalam mempresentasikan kebutuhan informasinya dalam bentuk *query* [SUT-10].

2.3.3. Teks *Preprocessing*

Berdasarkan ketidak teraturan struktur data teks, maka proses *text mining* memerlukan beberapa tahap awal yang pada intinya adalah mempersiapkan agar teks dapat diubah menjadi lebih terstruktur. Salah satu implementasi dari *text mining* adalah tahap *preprocessing text*. Tahap *preprocessing* adalah tahapan dimana aplikasi melakukan seleksi data yang akan diproses pada setiap dokumen. Proses *preprocessing* ini meliputi *case folding*, *tokenizing*, *filtering*, dan *stemming*. Gambar 2.1 adalah tahap dari *preprocessing*.



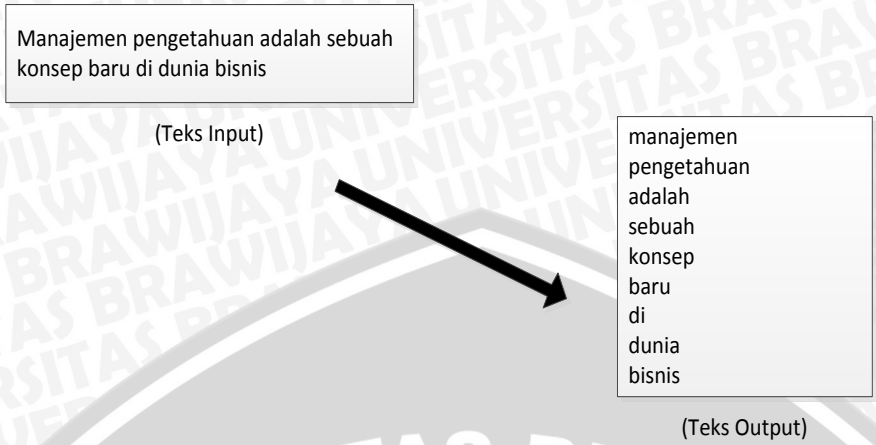
Gambar 2.1 Tahap *preprocessing*

Sumber : [NUG-11]

2.3.3.1. *Case Folding* dan *Tokenizing*

Tidak semua dokumen teks konsisten dalam penggunaan huruf kapital. Oleh karena itu, peran *case folding* dibutuhkan dalam mengkonversi keseluruhan teks dalam dokumen menjadi suatu bentuk standar (biasanya huruf kecil atau *lowercase*). Hanya huruf ‘a’ sampai dengan ‘z’ yang diterima. Karakter selain huruf dihilangkan dan dianggap delimiter. Delimiter pada sistem ini adalah semua tanda baca yang diantaranya '.', ',', "", """, '-', '/', '{', '}', '+', '_', '!', '@', '#', '$', '%', '^', '&', '*', '(', ')', '?', '>', '<', '[', ']', '|', '`', '~', ';', ':', '=', '\\', "\n", "\r".

Tahap *tokenizing / parsing* adalah tahap pemotongan string input berdasarkan tiap kata yang menyusunnya. Contoh dari tahap ini dapat dilihat pada gambar 2.2.



Gambar 2.2 Tahap *case folding* dan *tokenizing*

Sumber : [NUG-11]

Tokenisasi secara garis besar memecah sekumpulan karakter dalam suatu teks ke dalam satuan kata. Bagaimana membedakan karakter-karakter tertentu yang dapat diperlakukan sebagai pemisah kata atau bukan.

Sebagai contoh karakter whitespace, seperti enter, tabulasi, spasi dianggap sebagai pemisah kata. Namun untuk karakter petik tunggal ('), titik (.), semikolon (;), titk dua (:) atau lainnya, dapat memiliki peran yang cukup banyak sebagai pemisah kata.

Dalam memperlakukan karakter-karakter dalam teks sangat tergantung sekali pada kontek aplikasi yang dikembangkan. Pekerjaan *tokenisasi* ini akan semakin sulit jika juga harus memperhatikan struktur bahasa(*grammatikal*).

2.3.3.2. Filtering

Tahap *filtering* adalah tahap mengambil kata - kata penting dari hasil token. Bisa menggunakan algoritma *stoplist* (membuang kata yang kurang penting) atau *wordlist* (menyimpan kata penting). *Stoplist / stopword* adalah kata-kata yang tidak deskriptif yang dapat dibuang dalam pendekatan *bag-of-words*. Contoh stopwords adalah “yang”, “dan”, “di”, “dari” dan seterusnya. Contoh dari tahapan ini dapat dilihat pada gambar 2.3.

Gambar 2.3 Tahap *filtering*

Sumber : [NUG-11]

Kata-kata seperti "dari", "yang", "di", dan "ke" adalah beberapa contoh kata-kata yang berfrekuensi tinggi dan dapat ditemukan hampir dalam setiap dokumen (disebut sebagai *stopword*). Penghilangan *stopword* ini dapat mengurangi ukuran *index* dan waktu pemrosesan. Selain itu, juga dapat mengurangi level *noise*. Namun terkadang *stopping* tidak selalu meningkatkan nilai *retrieval*. Pembangunan daftar *stopword* (disebut *stoplist*) yang kurang hati-hati dapat memperburuk kinerja sistem IR. Belum ada suatu kesimpulan pasti bahwa penggunaan *stopping* akan selalu meningkatkan nilai *retrieval*, karena pada beberapa penelitian, hasil yang didapatkan cenderung bervariasi [MAH-08].

2.3.3.3. Stemming

Pembuatan indeks dilakukan karena suatu dokumen tidak dapat dikenali langsung oleh suatu sistem temu kembali informasi atau *information retrieval (IR)* sistem. Oleh karena itu, dokumen tersebut terlebih dahulu perlu dipetakan ke dalam suatu representasi dengan menggunakan teks yang berada di dalamnya. Teknik *stemming* diperlukan selain untuk memperkecil jumlah indeks yang berbeda dari suatu dokumen, juga untuk melakukan pengelompokan kata-kata lain yang memiliki kata dasar dan arti yang serupa namun memiliki bentuk atau *form* yang berbeda karena mendapatkan imbuhan yang berbeda. Sebagai contoh kata berjalan, menjalankan, perjalanan, akan distem ke *root word*-nya yaitu "jalan". Namun, seperti halnya *stopping*, kinerja *stemming* juga bervariasi dan sering tergantung pada domain bahasa yang digunakan. Proses *stemming* pada teks

berbahasa Indonesia berbeda dengan *stemming* pada teks berbahasa Inggris. Pada teks berbahasa Inggris, proses yang diperlukan hanya proses menghilangkan *sufiks*. Sedangkan pada teks berbahasa Indonesia selain *sufiks*, *prefiks*, dan *konfiks* juga dihilangkan. Contoh dari tahapan ini pada teks adalah seperti pada gambar 2.4 [AGU-09].

Gambar 2.4 Tahap Stemming

Sumber : [NUG-11]

2.4. Algoritma *Stemming* Arifin

Algoritma ini didahului dengan pembacaan tiap kata dari file sampel. Sehingga input dari algoritma ini adalah sebuah kata yang kemudian dilakukan :

1) Pemeriksaan semua kemungkinan bentuk kata. Setiap kata diasumsikan memiliki 2 Awalan *(prefiks)* dan 3 Akhiran *(sufiks)*.Sehingga bentuknya menjadi :

   Prefiks 1 + *Prefiks* 2 + Kata dasar + *Sufiks* 3 + *Sufiks* 2 + *Sufiks* 1

Seandainya kata tersebut tidak memiliki imbuhan sebanyak imbuhan di atas, maka imbuhan yang kosong diberi tanda x untuk *prefiks* dan diberi tanda xx untuk *sufiks*.

2) Pemotongan dilakukan secara berurutan sebagai berikut :

   AW : AW (Awalan)

   AK : AK (Akhiran)

   KD : KD (Kata Dasar)

   a. AW I, hasilnya disimpan pada p1 (*prefiks* 1)

   b. AW II, hasilnya disimpan pada p2 (*prefiks* 2)

c. AK I, hasilnya disimpan pada s1 (*sufiks* 1)

d. AK II, hasilnya disimpan pada s2 (*sufiks* 2)

e. AK III, hasilnya disimpan pada s3 (*sufiks* 3)

Pada setiap tahap pemotongan di atas diikuti dengan pemeriksaan di kamus apakah hasil pemotongan itu sudah berada dalam bentuk dasar. Kalau pemeriksaan ini berhasil maka proses dinyatakan selesai dan tidak perlu melanjutkan proses pemotongan imbuhan lainnya. Contoh pemenggalan kata "mempermainkannya"

a. Langkah 1 :

Cek apakah kata ada dalam kamus

Ya : Success

Tidak : lakukan pemotongan AW I

Kata = permainkannya

b. Langkah 2 :

Cek apakah kata ada dalam kamus

Ya : Success

Tidak : lakukan pemotongan AW II

Kata = mainkannya

c. Langkah 3 :

Cek apakah kata ada dalam kamus

Ya : Success

Tidak : lakukan pemotongan AK I

Kata = mainkan

d. Langkah 4 :

Cek apakah kata ada dalam kamus

Ya : Success

Tidak : lakukan pemotongan AK II

Kata = main

e. Langkah 5 :

Cek apakah kata ada dalam kamus

Ya : Success

Tidak : lakukan pemotongan AK III. Dalam

hal ini AK III tidak ada, sehingga kata tidak diubah.

Kata = main

f. Langkah 6

Cek apakah kata ada dalam kamus

Ya : Success

Tidak : "Kata tidak ditemukan"

3) Namun jika sampai pada pemotongan AK III, belum juga ditemukan di kamus, maka dilakukan proses kombinasi. KD yang dihasilkan dikombinasikan dengan imbuhan-imbuhannya dalam 12 konfigurasi berikut :

a. KD
b. KD + AK III
c. KD + AK III + AK II
d. KD + AK III + AK II + AK I
e. AW I + AW II + KD
f. AW I + AW II + KD + AK III
g. AW I + AW II + KD + AK III + AK II
h. AW I + AW II + KD + AK III + AK II + AK I
i. AW II + KD
j. AW II + KD + AK III
k. AW II + KD + AK III + AK II
l. AW II + KD + AK III + AK II + AK I

Sebenarnya kombinasi a, b, c, d, h, dan l sudah diperiksa pada tahap sebelumnya, karena kombinasi ini adalah hasil pemotongan bertahap tersebut. Dengan demikian, kombinasi yang masih perlu dilakukan tinggal 6 yakni pada kombinasi-kombinasi yang belum dilakukan (e, f, g, i, j, dan k). Tentunya bila hasil pemeriksaan suatu kombinasi adalah ‘ada’, maka pemeriksaan pada kombinasi lainnya sudah tidak diperlukan lagi.

Pemeriksaan 12 kombinasi ini diperlukan, karena adanya fenomena *overstemming* pada algoritma pemotongan imbuhan. Kelemahan ini berakibat

pada pemotongan bagian kata yang sebenarnya adalah milik kata dasar itu sendiri yang kebetulan mirip dengan salah satu jenis imbuhan yang ada. Dengan 12 kombinasi itu, pemotongan yang sudah terlanjur tersebut dapat dikembalikan sesuai posisinya. [ARI-02]

2.5. *String Matching* (Pencocokan Teks)

2.5.1. Pengertian *String Matching*

String matching atau pencocokan *string* adalah suatu metode yang digunakan untuk menemukan suatu keakuratan atau hasil dari satu atau beberapa pola teks yang diberikan. *String matching* merupakan pokok bahasan yang penting dalam ilmu komputer karena teks merupakan bentuk utama dari pertukaran informasi antar manusia, misalnya pada literatur, karya ilmiah, halaman web dsb [NUG-11].

String matching digunakan dalam lingkup yang bermacam-macam, misalnya pada pencarian dokumen, pencocokan DNA *sequences* yang direpresentasikan dalam bentuk string dan juga *string matching* dapat dimanfaatkan untuk mendeteksi adanya plagiarisme dalam karya seseorang. *String-matching* fokus pada pencarian satu, atau lebih umum, semua kehadiran sebuah kata (lebih umum disebut *pattern*) dalam sebuah teks. Semua algoritma yang akan dibahas mengeluarkan semua kehadiran pola dalam teks. Pola dinotasikan sebagai x = x[0..m-1]; m. Teks dinotasikan sebagai y = y[0..n-1]; n. Kedua string dibentuk dari set karakter yang disebut alphabet dinotasikan Σ dengan ukuran σ [AMO-06].

Permasalahan pencocokan string (*string matching*) merupakan permasalahan yang sangat terkenal dalam dunia informatika. Contoh implementasi dari permasalahan pencocokan string adalah pada pencocokan sebuah string pada Microsoft Word atau editor, atau dalam kasus yang lebih besar lagi, yaitu pencocokan website dengan memasukkan kata-kata kunci sebagaimana yang telah diimplementasikan pada search engine, seperti Yahoo atau Google. Masalah utama dalam pencarian string adalah untuk mencari sebuah string yang terdiri dari beberapa karakter (yang biasa disebut pattern) dalam sejumlah besar text. Pencarian string juga bisa digunakan untuk mencari pola bit dalam sejumlah besar file *binary*.

2.5.2. Algoritma Levenshtein Distance

Algoritma *Levenshtein* dinamakan sesuai penemu algoritma tersebut yaitu Vladimir Levenshtein. Dia menemukan algoritma tersebuat pada tahun 1965. *Levenshtein distance* merupakan *matriks* yang digunakan untuk mengukur keterbedaan jarak antara dua sekuens. Perhitungan *edit distance* didapatkan dari *matriks* yang digunakan untuk menghitung jumlah perbedaan antara dua string. Perhitungan jarak antara dua string ini ditentukan dari jumlah minimum operasi perubahan untuk membuat string A menjadi string B. Levenshtein distance antara dua string ditentukan berdasarkan jumlah minimum perubahan/pengeditan yang diperlukan untuk melakukan tranformasi dari satu bentuk string ke bentuk string yang lain. Misalnya jika source string (s) adalah “hullo” dan target string (t) adalah “hallo” maka levenshtein distance (LD) = 1, hal tersebut berarti dibutuhkan sebuah operasi (subtitution) untuk mengubah source string menjadi sama dengan target string. Operasi yang dilakukan dan diperbolehkan digunakan dalam menentukan levenshtein distance ini ada 3 macam operasi yaitu: [AND-11]

1. *Insertion* (penyisipan)

Insertion atau penyisipan adalah operasi menyisipkan sebuah karakter kedalam string tertentu. Contohnya menyisipkan sebuah karakter ‘a’ kedalam string “bca” tepat setelah karakter ‘b’, maka string “bca” berubah menjadi “baca” setelah dilakukan operasi insertion. String bca dimisalkan T sedangkan string baca dimisalkan S. Prosesnya ditunjukan seperti berikut

	1	2	3	4
T =	b	c	a	
S =	b	-	c	a
		a		

2. *Deletion* (penghapusan)

Deletion adalah operasi melakukan penghilangan atau penghapusan sebuah karakter tertentu dari sebuah string. Misalnya, menghapus karakter ‘m’ pada string “senam”. Setelah dilakukan operasi deletion, string akan menjadi “sena”.

String senam dimisalkan T untuk string sena dimisalkan S. Prosesnya ditunjukk an seperti berikut.

	1	2	3	4	5
T =	s	e	n	a	m
S =	s	e	n	a	-

3. *Substitution* (penukaran)

Substitution adalah operasi menukarkan sebuah karakter pada string tertentu dengan karakter lain. Misalnya, menukarkan karakter ‘b’ pada string “bakar” dengan karakter baru ‘p’. setelah dilakukan substitution, string akan menjadi “pakar”. String bakar dimisalkan T sedangkan string bakar dimisalkan S. Prosesnya ditunjukkan sebagai berikut.

	1	2	3	4	5
T =	b	a	k	a	r
S =	p	a	k	a	r
	b				

Algoritma ini menghitung kemiripan antar kata berdasarkan total biaya terkecil dari transformasi salah satu kata menjadi kata yang lain dengan menggunakan edit-rules, yaitu penambahan karakter *(insertion),* penggantian karakter *(subtituio*n), dan penghapusan karakter *(deletion).* Sebagai contoh hasil penggunaan algoritma ini, string “anna” dan “anni” memiliki distance 1 karena untuk mengubah string “anna” menjadi anni hanya diperlukan satu operasi saja. Dalam kasus dua string diatas, string “anna” dapat menjadi “anni” hanya dengan melakukan satu operasi *substitution* karakter ‘a’ dengan karakter ‘i’ pada string “anna”. Prosesnya ditunjukkan sebagai berikut.

	1	2	3	4
T =	a	n	n	a
S =	a	n	n	i
				a

2.5.3. Mekanisme Perolehan Nilai *Distance* pada *Algoritma Levenshtein Distance*

Algoritma ini berjalan mulai dari pojok kiri atas sebuah array dua dimensi yang telah diisi sejumlah karakter sring awal dan string target dan diberikan nilai *cost*. Nilai cost pada ujung kanan bawah menjadi nilai edit distance yang menggambarkan jumlah perbedaan dua *string*. Berikut adalah algoritma *levenshtein* yang digunakan untuk mencari nilai *distance* antara dua masukan *string*, dengan m adalah panjang dari *string* pertama, dan n adalah panjang *string* kedua.

```
Function LevDistance(input s:string[1..m],
                     t: string[1..n])
Deklarasi
i,j = integer;
d[0..m, 0..n] = integer;
Algoritma
for i from 0 to m {perbandingan dengan kosong}
    d[i,0] = i;
for j from 0 to n {perbandingan dengan kosong}
    d[0,j] = j;
for j from 1 to n{
    for i from 1 to m{
        if s[i] = t[j] then
            d[i,j] = d[i-1, j-1]
        else
            d[i,j] = minimum(
                        d[i-1,j-1]+1,
                        d[i-1,j]+1,
                        d[i,j-1]+1
                    )
    }
}
return d[m,n]
```

Kode Sumber 2.1 *Pseudocode* Algoritma *Levenshtein Distance*

Fungsi *levenshtein distance* pada Kode Sumber 2.1 menggunakan masukan dua buah string dan menghasilkan nilai *"distance"*-nya. Seperti yang telah disebutkan pada bagian sebelumnya, nilai *"distance"* merupakan nilai yang menunjukkan jumlah modifikasi minimum yang harus dilakukan untuk melakukan pengubahan string yang satu ke string yang lain. Untuk lebih jelasnya perhatikan matrik ilustrasi pencari levenshtein distance antara dua string berikut yaitu "penjara" dan "jarak".

Jika kita melihat sekilas, kedua string "penjara" dan "jarak" memiliki jarak 4. Berarti untuk mengubah string "jarak" menjadi "penjara" diperlukan 4 operasi yaitu:

1. Menyisipkan karakter 'p'
   jarak → pjarak
2. Menyisipkan karakter 'e'
   pjarak → pejarak
3. Menyisipkan karakter 'n'
   pejarak → penjarak
4. Menghapus karakter 'k'
   penjarak → penjara

Dengan menggunakan representasi matrik dapat ditunjukkan dengan tabel berikut:

Tabel 2.1 Matrik *Levenshtein Distance*

		p	e	n	j	a	r	a
	0	1	2	3	4	5	6	7
j	1							
a	2							
r	3							
a	4							
k	5							

Sumber : [Rancangan]

Pada tabel diatas, elemen baris 1 kolom 1 (M[1,1]) adalah jumlah operasi yang diperlukan untuk mengubah substring dari kata jarak yang diambil mulai dari karakter awal sebanyak 1 (j) ke substring dari kata penjara yang diambil mulai dari karakter awal sebanyak 1 (p). Sementara elemen M [2,5] adalah jumlah operasi antara 'ja' (substring dari kata jarak yang diambil mulai dari karakter awal sebanyak 2) dengan 'penja' (substring dari kata penjara yang diambil mulai dari karakter awal sebanyak 5). Dari ilustrasi diatas dapat disimpulkan bahwa elemen M [p,q] adalah jumlah operasi antara substring kata pertama yang diambil mulai dari awal sebanyak p dengan substring kata kedua yang diambil dari awal sebanyak q. Sehingga dengan peraturan tersebut matrik pada Tabel 2.1 dapat diisi sebagai berikut:

Tabel 2.2 Matrik *Levenshtein Distance* kata jarak dan penjara

		p	e	n	j	a	r	a
	0	1	2	3	4	5	6	7
j	1	1	2	3	3	4	5	6
a	2	2	2	3	4	3	4	5
r	3	3	3	3	4	4	3	4
a	4	4	4	4	4	4	4	3
k	5	5	5	5	5	5	5	4

Sumber : [Rancangan]

Pada tabel 2.2 merupakan matrik yang menunjukkan bagaimana proses pencarian nilai *"distance"* untuk string "jarak" dan "penjara". Pengecekan dimulai atau diiterasi dari awal dari kedua string yaitu karakter-karakter paling awal dari kedua string, lalu dilakukan operasi-operasi tertentu pada string seperti insertion, deletion, atau substitution sampai semua karakter dibandingkan. Nilai *"distance"* akhir adalah yang terdapat pada akhir atau ujung kedua string yang dibandingkan yaitu pada bagian pojok kanan bawah pada matrik. Dari proses pencarian nilai *"distance"*-nya, didapat bahwa jumlah modifikasi minimum yang dilakukan untuk mengubah string "jarak" menjadi string "penjara" adalah sebanyak 4 operasi yaitu: penyisipan karakter 'p', penyisipan karakter 'e', penyisipan karakter 'n', dan deletion karakter 'k'.

2.5.4. Menghitung *Similarity* dengan *Algoritma Levenshtein Distance*

Sebelum dilakukan perhitungan *similarity* menggunakan *levenshtein distance* sebelumnya jawaban beserta kunci jawaban sudah melalui *pre-processing.* Langkah-langkah dalam menentukan *similarity,* dimisalkan *str1* adalah string pertama dan *str2* adalah string kedua. Setelah dilakukan penghitungan dari kedua string tersebut menggunakan *levenshtein distance*, maka algoritma ini akan memberikan angka sebagai perbedaan dari kedua string, dimisalkan *Distance* adalah *edit distance.*

Setelah mendapatkan biaya edit-distance maka untuk menghitung nilai Levenshtein Distance atau perhitungan similarity menggunakan Persamaan berikut [DAN-06].

$$Similarity = \left\{1 - \frac{edit\ distance}{maxLength(str1,str2)}\right\} * 100 \quad (2.1)$$

Nilai kemiripan (similarity score) diasumsikan pada rentang 0 (nol) hingga 1 (satu), yang artinya nilai 1 adalah nilai maksimum yang menunjukan bahwa dua kata adalah sama identik. Pendekatan yang digunakan oleh penelitian ini mampu mengukur nilai kemiripan antar dua string berdasarkan pada susunan karakter.

Pada tabel 2.2 didapatkan nilai edit distance 4 kemudian *maxLength* antara string 1 dan string 2 adalah 7 maka akan didapatkan *Levenshtein Distance* dari kata penjara dengan jarak adalah:

$$\begin{aligned} Similarity &= \left(1 - \frac{4}{7}\right) * 100 \\ &= \left(\frac{7}{7} - \frac{4}{7}\right) * 100 \\ &= \left(\frac{3}{7}\right) * 100 \\ &= 43 \end{aligned}$$

Semakin besar nilai jarak edit, semakin rendah tingkat kemiripan antara kedua string (judul). Besar dan kecil nilai jarak edit menjadi relatif dalam persentase kemiripan terhadap string yang lain. Maka nilai diatas kesamaan kata antara penjara dengan jarak 43% hasil tersebut dapat dijadikan acuan sebagai penilaian antara kunci jawaban dengan jawaban.