BAB 2
LANDASAN TEORI

2.1. *Typographical Error*

Typographical error atau sering disingkat menjadi *typo* adalah kesalahan yang dibuat dalam proses pengetikan seperti ejaan atau meninggalkan suatu kata. Salah satu yang paling umum dari kesalahan ketik atau *typo* adalah transposisi huruf, dimana semua huruf yang diperlukan untuk mengeja kata yang hadir, tetapi dalam urutan yang salah [8]. Ini jenis kesalahan yang sering terjadi ketika seseorang mengetik dengan cepat dan tidak cukup memperhatikan apa yang ditulis. Juga tidak jarang melihat substitusi huruf disebabkan oleh slip jari pada keyboard. Kesalahan tersebut dapat terjadi karena kegagalan mekanis atau slip dari tangan atau jari. Contoh-contoh *typo* seperti, duplikasi sederhana, ketertinggalan, dan transposisi atau tertukarnya posisi huruf [8]. Jenis-jenis *typo* yang sering terjadi dalam pengetikan diantaranya kekurangan salah satu huruf, kelebihan, dan kesalahan penempatan [9]. Contoh kekurangan huruf adalah seperti kata ‘nasi’ menjadi ‘nsi’ [8]. Contoh kelebihan huruf seperti kata ‘nasi’ menjadi ‘nasii’, dan contoh kesalahan penempatan huruf seperti kata ‘nasi’ menjadi kata ‘nsai’.

2.2. Algoritma *Levenshtein*

Algoritma *Levenshtein* merupakan algoritma pencarian jumlah perbedaan string yang ditemukan oleh Vladimir Levenshtein, seorang ilmuwan Rusia, pada tahun 1965 [10]. Algoritma ini digunakan secara luas dalam berbagai bidang, misalnya mesin pencari, pengecek ejaan, pengenalan pembicaraan, pengucapan dialek, analisis DNA, dan lain-lain. Dua buah string dibandingkan untuk mengetahui nilai edit distance-nya. Edit distance adalah suatu pengukuran (metrik) yang dihasilkan melalui perhitungan jumlah perbedaan yang terdapat pada dua string. *Edit-distance* antara dua string didefinisikan sebagai jumlah minimum perubahan yang diperlukan untuk mengganti suatu string dengan string lain, dengan operasi penambahan (*insert*), operasi penggantian (*replace*), operasi

penukaran (*transpose*), operasi penghapusan (*delete*). Algoritma ini menggunakan matriks dua dimensi dalam membandingkan kedua string. Algoritma ini berjalan mulai dari pojok kiri atas hingga ke pojok kanan bawah. Seluruh baris pertama diisi dengan nilai berurut mulai dari 0 hingga n begitu juga dengan kolom pertama. Cara pengisiannya dapat dilihat pada Gambar 2.1 Inisialisasi Awal Tabel *Levenshtein*.

		T	O	P
	0	1	2	3
T	1			
O	2			
P	3			
I	4			

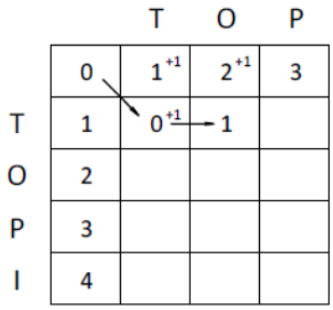
Gambar 2. 1 Inisialisasi Awal Tabel *Levenshtein*

Pada gambar diatas kata “top” akan dilakukan perbandingan dengan kata “topi” untuk melihat nilai edit distance-nya. Pengisian dilakukan mulai dari blok (1,1) dengan membandingkan karakter yang ada pada blok (0,1) dan blok (1,0). Apabila karakter yang dibandingkan sama, turunkan nilai yang ada pada blok (0,0) yaitu angka 0. Nilai *edit distance*-nya dapat dilihat pada Gambar 2.2 Penurunan Angka 0 ke Baris 1 Kolom 1.

		T	O	P
	0	1	2	3
T	1	0		
O	2			
P	3			
I	4			

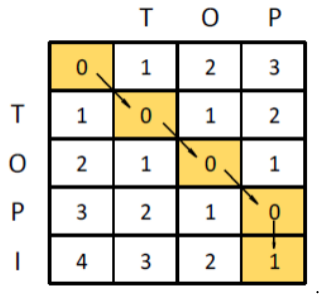
Gambar 2. 2 Penurunan Angka 0 ke Baris 1 Kolom 1

Pengisian blok berikutnya dilakukan dengan membandingkan karakter dengan baris yang sama. Apabila karakter yang dibandingkan tidak sama, maka nilai blok (0,2) ditambahkan dengan angka 1. Sama halnya dengan blok (1,1) dan blok (0,1). Setelah itu lakukan perbandingan nilai dari blok (0,2), (1,1), dan (0,1) untuk mencari nilai minimumnya. Selanjutnya simpan pada blok (1,2). Hasilnya dapat dilihat pada Gambar 2.3 Pengisian Blok (1,2).



Gambar 2. 3 Pengisian Blok (1,2)

Langkah selanjutnya isi semua blok sampai akhir dengan langkah yang sama. Hasil akhir dari pengisian tabel Levenshtein diatas dapat dilihat pada Gambar 2.4 Hasil Akhir Pengisian Tabel *Levenshtein*.



Gambar 2. 4 Hasil Akhir Pengisian Tabel *Levenshtein*

Pada pendekatan secara tradisional, jika ini diimplementasikan di kasus typo checking maka kata yang akan dicek akan di hitung nilai *edit-distance* nya dengan setiap kata yang ada di kamus lalu diurutkan berdasarkan nilai *edit-distance* terkecil sehingga pendekatan ini memerlukan waktu sangat lama [6].

2.3. Pendekatan Peter Norvig

Pada tahun 2007 Peter Norvig menemukan pendekatan lain[5]. Pendekatan Peter Norvig menghasilkan semua kemungkinan kata dengan semua operasi *edit-distance* yaitu operasi penambahan (*insert*), operasi penggantian (*replace*), operasi penukaran (*transpose*), operasi penghapusan (*delete*) dari kata yang terdeteksi *typo* dan mencarinya dalam kamus. Sebagai contoh kata **top** jika terdeteksi *typo* maka akan dibuat semua kemungkinan kata dari semua operasi *edit-distance*=1 dan hasilnya bisa dilihat pada Tabel 2.1 Hasil Pendekatan Peter Norvig.

Tabel 2. 1 Hasil Pendekatan Peter Norvig

Insert	Replace	Transpose	Delete
atop, btop, ctop, dtop, etop, ftop, gtop, htop, itop, jtop, ktop, ltop, mtop, ntop, otop, ptop, qtop, rtop, stop, ttop, utop, vtop, wtop, xtop, ytop, ztop, taop, tbop, tcop, tdop, teop, tfop, tgop, thop, tiop, tjop, tkop, tlop, tmop, tnop, toop, tpop, tqop, trop, tsop, ttop, tuop, tvop, twop, txop, tyop, tzop, toap, tobp, tocp, todp, toep, tofp, togp, tohp, toip, tojp, tokp, tolp, tomp, tonp, toop, topp, toqp, torp, tosp, totp, toup, tovp, towp, toxp, toyp, tozp, topa, topb, topc, topd, tope, topf, topg, toph, **topi**, topj, topk, topl, topm, topn, topo, topp, topq, topr, tops, topt, topu, topv, topw, topx, topy, topz	aop, bop, cop, dop, eop, fop, gop, hop, iop, jop, kop, lop, mop, nop, oop, pop, qop, rop, sop, top, uop, vop, wop, xop, yop, zop, tap, tbp, tcp, tdp, tep, tfp, tgp, thp, tip, tjp, tkp, tlp, tmp, tnp, top, tpp, tqp, trp, tsp, ttp, tup, tvp, twp, txp, typ, tzp, toa, tob, toc, tod, toe, tof, tog, toh, toi, toj, tok, tol, tom, ton, too, top, toq, tor, tos, tot, tou, tov, tow, tox, toy,	otp, tpo	op, tp, to

Insert	Replace	Transpose	Delete
	toz		

Untuk kata dengan panjang n, ukuran alfabet a, *edit-distance* d = 1, akan ada n penghapusan, n-1 penukaran, a * n penggantian, dan a*(n + 1) penambahan. Ini jauh lebih baik daripada pendekatan naif tetapi kata yang dihasilkan bisa sangat bannyak [6].

2.4. Algoritma *Symspell*

Pada tahun 2012 Wolf Garbe menemukan pendekatan lain yang lebih efisien, dan membuat waktu untuk pencarian rekomendasi kata yang terdeteksi *typo* menjadi lebih cepat, dan diberi nama *Symspell* (*Symmetric Delete Spelling Correction*) [6]. *SymSpell* (*Symmetric Delete Spelling Correction*) yang berbasis dari penelitian Peter Norvig dan berhasil meningkatkan performanya menjadi seribu kali lebih cepat. Algoritma *SymSpell* mengurangi kompleksitas dari pencarian kamus dengan melakukan pengghapusan saja. Berbeda dengan pendekatan Peter Norvig yang melakukan semua operasi *edit-distance* di kata yang akan dicek, pendekatan Wolf Garbe melakukan penghapusan saja namun tidak hanya pada kata yang akan dicek namun kata yang ada di kamus, namun penghapusna huruf yang ada di kamus hanya dilakukan sekali saja, sehingga bisa lebih cepat.

Secara sederhana algoritma *Sysmspell* terdiri dari lima tahap, yaitu:

1) Lakukan operasi penghapusan huruf yang ada di kamus lalu simpan hasilnya (nantinya disebut dataset).
2) Load hasil proses nomor 1.
3) Untuk setiap kata yang terdeteksi typo, lakukan operasi penghhapusan.
4) Cari setiap hasil penghapusan di dataset.
5) Jika ditemukan, maka tambahkan ke rekomendasi kata yang benar.

Sebagai contoh jika kata **hari** ada di kamus dan kata **yari** adalah kata yang terdeteksi *typo*. Kedua kata tersebut di lakukan operasi penghapusan *edit-distance*=1. Hasilnya bisa dilihat pada Tabel 2.2 Hasil Pendekatan Wolf Garbe.

Tabel 2. 2 Hasil Pendekatan Wolf Garbe

Hari	yari
Ari	**ari**
hri	yri
hai	yai
har	yar

Jika dilihat pada tabel diatas ada kesamaan kata **ari**, maka kata **hari** akan masuk ke daftar rekomendasi kata yang benar.

Kelemahan dari pendekatan ini adalah waktu penghapusan huruf yang ada di kamus dan tambahan penggnaan media penyimpanan karena hasil penghapusan tersebut disimpan di *file* agar bisa langsung digunakan ketika diperlukan (tidak membuat ulang). Menerapkan pra-perhitungan ke pendekatan Peter Norvig tidak akan layak karena harus melakukan semua operasi *edit-distance*, karenanya akan menghasilkan waktu yang sangat lama dan konsumsi media penyimpaan yang sangat besar.

Karena pendekatan ini dan pendekatan Peter Norvig tidak ada proses pengurutan berdasarkan prioritas seperti pendekatan secara tradisional yang diurutkan berdasarkan *score* yang didapat tiap kata di kamus. Sehingga urutan prioritas untuk pendekatan ini dilakukan dengan bantuan korpus.

Korpus adalah kumpulan teks alami, baik bahasa lisan maupun bahasa tulis, yang disusun secara sistematis. Dikatakan “alami” karena teks yang dikumpulkan

merupakan teks yang diproduksi dan digunakan secara wajar dan tidak dibuat-buat. Teks-teks tersebut termasuk novel, buku dan kertas akademis, koran, majalah, rekaman siaran pembicaraan dan wawancara, blog, jurnal daring, dan kelompok diskusi, dan banyak lagi. Dikatakan "sistematis" karena struktur dan isi korpus mengikuti prinsip ekstralinguistik tertentu, khususnya prinsip pengambilan sampel, yaitu prinsip dasar dalam pemilihan teks yang akan dimasukkan ke dalam korpus [11].

Pendekatan ini menghitung kemunculan setiap kata yang muncul di korpus dan mengurutkan hasil rekomendasi berdasarkan kemunculan kata tersebut karena dinilai kata tersebut lebih banyak ditulis. Karena jika dilihat dari contoh sebelumnya rekomendasi kata benar untuk kata **yari** bisa saja ada lebih dari satu, misal **hari**, **dari**, **tari**.

2.5. *Question Answering System*

Menurut KBBI, arti kata 'tanya' adalah 'permintaan keterangan', arti kata 'jawab' adalah 'sahut' atau 'balas', dan arti kata 'proses' adalah 'rangkaian tindakan, pembuatan, atau pengolahan yang menghasilkan produk'. Dari keterangan tersebut bisa disimpulkan bahwa 'proses tanya jawab' adalah rangkaian tindakan permintaan keterangan yang mendapatkan sahutan atau balasan. Dalam hal ini, sahutan atau balasan yang terjadi tentu saja harus berhubungan dengan permintaan keterangan yang ada.

Question Answering System merupakan sebuah sistem pada komputer yang berisi sebuah cara otomatis untuk menjawab pertanyaan menggunakan bahasa alami yang biasa dipakai manusia. Jawaban ini akan berisi informasi yang berasal dari sebuah sumber basis data, baik terstruktur maupun tidak terstruktur [12]. Dengan kata lain, sistem tanya jawab merupakan implementasi proses tanya jawab pada komputer.

2.6. *Metode Rule-Based*

Rule jika diterjemahkan secara literal berarti aturan. Disekitar kita banyak sekali terdapat aturan-aturan yang ada. Fungsi dari aturan-aturan tersebut adalah

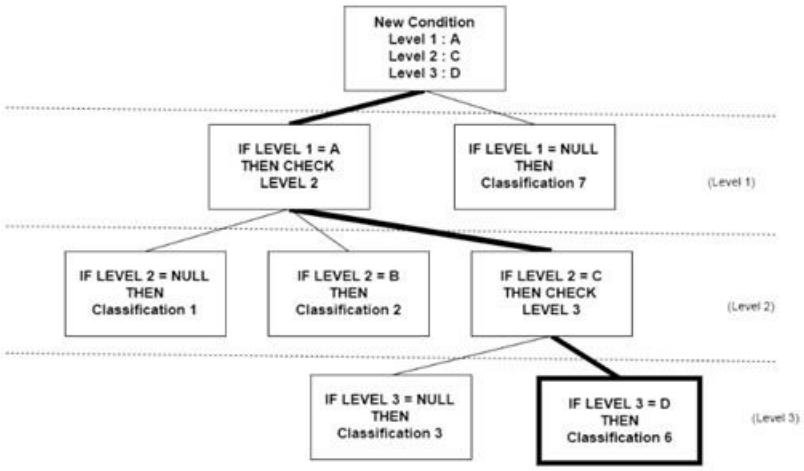
untuk membatasi apa saja yang dilakukan oleh manusia. Beberapa contoh aturan yang ada di kehidupan kita misalnya, hukum, aturan lalu lintas, dan aturan penggunaan sebuah peralatan. Di dalam dunia informatika, *Rule-Based System* akan memberikan sebuah alat yang berguna dalam pembangunan aplikasi-aplikasi tertentu. Menurut Antoni Ligeza [12], bentuk dasar dari rule-based system adalah sebagai berikut.

rule : (*preconditions*) -> (*conclusion*)

Dimana *preconditions* adalah sebuah formula yang mendefinisikan kapan rule atau aturan tersebut diaplikasikan, dan *conclusion* adalah definisi dari efek dari pengaplikasian rule atau aturan tersebut [13]. Beberapa contoh metode *rule-based* yang banyak digunakan antara lain *Ripple Down Rules* (RDR), *Multiple Classification Ripple Down Rules* (MCRDR), dan *Decision Tree*.

2.6.1 *Multiple Classification Ripple Down Rules* (MCRDR)

Pada penelitian ini metode *rule-based* yang digunakan adalah metode *Multiple Classification Ripple Down Rules* (MCRDR). MCRDR merupakan ekstensi dari RDR, dimana MCRDR adalah sebuah metode akuisisi informasi yang memungkinkan seorang ahli melakukan baik proses akuisisi informasi, ataupun proses maintenance pada sebuah sistem pakar [14]. Penggunaan MCRDR ditujukan untuk menutupi kelemahan RDR dimana RDR hanya cocok digunakan untuk masalah yang memerlukan klasifikasi tunggal atau hanya memiliki satu kategori solusi. Pertanyaan yang dimasukkan oleh pengguna akan sangat mungkin memiliki lebih dari satu kategori solusi untuk setiap pertanyaannya. Contohnya pada pertanyaan ‘berapa dan kapan’, pertanyaan tersebut akan memiliki dua kategori jawaban, jawaban untuk kata tanya berapa dan jawaban untuk kata tanya kapan. Gambar 2.5 menunjukkan skema *tree* dari sebuah MCRDR.



Gambar 2. 5 Skema Tree

2.7. UML *(Unified Modeling Language)*

UML adalah bahasa yang dimaksudkan untuk memodelkan aplikasi komputer [16]. Salah satu tujuan dari UML adalah untuk menyediakan lingkungan *develompment* dengan bahasa desain yang stabil dan umum yang dapat digunakan untuk mengembangkan dan membangun aplikasi komputer. Dengan menggunakan UML, *developer* bisa membaca dan mendistribusikan struktur sistem dan rencana desain aplikasi dengan lebih mudah. UML menyediakan beberapa jenis diagram yang meningkatkan kemudahan memahami aplikasi dalam proses *development*. Beberapa diagram yang digunakan pada penelitian ini antara lain:

1) *Use-case diagram*

Use-case diagram menggambarkan unit fungsionalitas yang disediakan oleh sistem. Tujuan utama dari *use-case diagram* adalah untuk membantu memvisualisasikan kebutuhan fungsional dari sistem, termasuk hubungan "aktor" dengan *use-case*, serta hubungan antara *use-case* yang berbeda [15][16].

2) *Activity diagram*

Activity diagram menunjukkan aliran kontrol antara dua atau lebih abjek saat memproses suatu kegiata. *Activity diagram* dapat digunakan untuk memodelkan tindakan objek pada *higher-level processes*, karena *activity diagram* yang "kurang teknis" jika dibandingkan dengan sequence diagram [15][16].

3) *Class diagram*

Class diagram menunjukkan bagaimana entitas yang berbeda berhubungan satu sama lain. *Class diagram* menampilkan struktur statis dari sistem. *Class diagram* juga menunjukan implementasi dari *class* yang ada di sistem [15][16].

4) *Sequence diagram*

Sequence diagram menunjukkan aliran rinci untuk *use-case* tertentu atau bahkan hanya bagian dari *use-case* tertentu. *Sequence diagram* menunjukkan hubungan antara objek yang berbeda dalam urutan, dan dapat menunjukkan secara rinci urutan pemanggilan kepada objek yang berbeda [15][16].

2.8. *Java programming language*

Java programming language adalah bahasa pemprograman berorientasi objek dan berbasis *class* untuk keperluan umun dan dirancang cukup sederhana untuk *programmer* [17]. *Java programming language* tergolong bahasa pemrograman tingkat tinggi dimana representasi mesin tidak tersedia melalui bahasa. *Java programming language* memiliki manajemen penyimpanan otomatis untuk menghindari masalah kemanan dealokasi secara eksplisit. *Java programming language* mendukung pembatasan akses eksternal ke member dari *packages*, *classes*, dan *interfaces*. Hal ini membantu dalam menulis program dengan skala besar.

Java programming language disusun dengan instruksi *bytecode* dan *binary format* yang didefinisikan dalam *Java Virtual Machine* (JVM). Program yang

dibuat dengan *Java programming language* berjalan di *Java Runtime EnvironmentI* (JRE), sehingga kita tidak perlu membuat program ulang atau melakukan kompilasi ulang untuk sistem operasi yang berbeda. Sistem operasi yang didukung antara lain *Linux, BSD, Mac OS X*, *Solaris* dan *Windows*.