Analisis Perbandingan Metode *Burkhard Keller Tree* dan *SymSpell* dalam *Spell Correction* Bahasa Indonesia

Muhammad Hafizh Ferdiansyah[1], I Kadek Dwi Nuryana[2]

[1,2] Jurusan Teknik Informatika, Prodi Teknik Informatika, Fakultas Teknik, Universitas Negeri Surabaya
[1]muhammadhafizh.18058@mhs.unesa.ac.id
[2]dwinuryana@unesa.ac.id

***Abstrak*— Dalam pembuatan sistem *spell correction* banyak faktor yang perlu diperhatikan untuk membuat sistem yang efektif dan berkualitas, salah satunya adalah kecepatan dan kebutuhan sistem. Beberapa metode dapat diterapkan untuk membuat sistem ini. Salah satu metode yang sering ditemui adalah *Burkhar Keller Tree* atau *BK Tree*. *BK Tree* merupakan metode populer yang digunakan dalam sistem *spell correction* karena kemudahannya, kemudian ada metode *Symmetric Delete Spelling Correction* atau *SymSpell* yang dikatakan memiliki kinerja yang sangat baik. Penelitian ini dilakukan untuk menguji dan menganalisa kinerja dari kedua metode tersebut sebagai sistem *spell correction* untuk Bahasa Indonesia. Hasil penelitian dengan kamus berisi 1.597.416 kosakata membuktikan metode *BK Tree* memiliki kinerja kecepatan yang lebih rendah dimana dalam pengujian catatan waktu tertinggi metode ini menyentuh 52 detik, namun metode memiliki kebutuhan sistem yang lebih kecil. Sedangkan metode *SymSpell* memiliki kinerja kecapatan yang jauh lebih cepat dengan nilai catatan waktu dalam pengujian tertinggi adalah 0.05 detik, namun kebutuhan sistem yang lebih besar. Pada pengujian akurasi didapatkan bahwa kedua metode memiliki hasil yang sama dengan rata-rata nilai *accuracy*, *precision*, dan *recall* secara berurutan sebesar 0,95, 0,89, dan 0,73. Dalam penelitian juga diketahui pentingnya sumber corpus yang digunakan untuk menyusun kamus, dimana penggunaan corpus yang bersumber dari Wikipedia Indonesia yang digunakan dalam penelitian ini masih kurang tepat karena masih ditemukannya kata-kata yang salah dalam penulisan dalam corpus tersebut.**

***Kata Kunci*— NLP, Spell Correction, BK Tree, SymSpell, Corpus**

I. PENDAHULUAN

Spelling correction adalah sebuah proses mendeteksi dan membenarkan kesalahan ejaan atau penulisan dalam sebuah teks [1]. Hal ini merupakan sebuah proses penting didalam natural language processing (NLP). *Spelling correction* memindai teks dan mengekstrak kata-kata yang terkandung didalamnya kemudian membandingkan tiap kata dengan daftar kata di dalam kamus utama yang digunakan sebagai pemeriksa ejaan. Ketika sebuah kata tidak ditemukan di dalam kamus kata tersebut akan dinyatakan sebagai kesalahan, untuk membenarkannya program akan mencari kata yang paling mirip dengan kata yang salah sebelumnya di dalam kamus [2]. Spelling Correction telah banyak digunakan, seperti pada mesin pencarian Google [3].

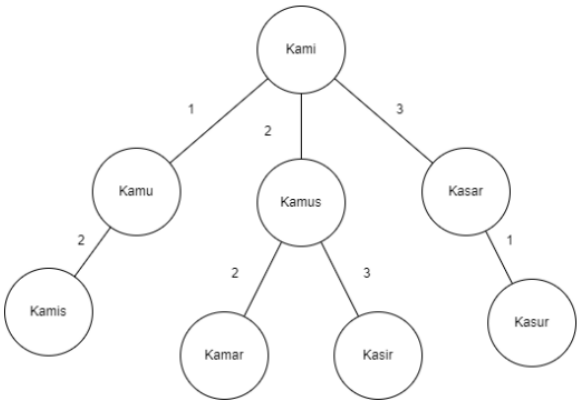
Dengan banyaknya kosa kata yang ada dalam setiap bahasa, untuk melakukan perbandingan tentunya kecepatan menjadi salah satu faktor penting yang harus diperhatikan dalam membuat sebuah spell correction [4]. Berbagai metode telah diterapkan dan dikembangkan untuk menghasilkan sistem yang baik. Salah satu metode yang paling populer dan dianggap paling cocok diterapkan pada *spell correction* yaitu *Burkhard Keller Tree* atau *BK Tree* [5]. *BK Tree* adalah sebuah struktur data berbasis *tree* yang digunakan untuk mengecek ejaan dan pencarian perkiraan string berdasarkan *edit distance* [6]. Untuk menghitung *edit distance* salah satu metode yang sering digunakan adalah Levenshtein Distance, metode ini dapat menghitung kemiripan dari dua buah string dengan menghitung nilai terkecil dari proses substitusi, penyisipan, dan penghapusan yang diperlukan untuk mengubah kata satu menjadi kata lainnya [7].

Metode *spell correction* lainnya yaitu SymSpell atau *Symmetric Delete Spelling Correction*, yang menurut penciptanya bisa melakukan spell correction jauh lebih cepat dari metode-metode lain. Dibandingkan dengan metode yang ada sebelumnya seperti *BK Tree* yang mencoba menemukan entri kamus dengan jarak edit terkecil dari ketentuan *query*, *SymSpell* menggunakan cara yang berbeda utnuk mencari kata dalam kamus, sehingga menghasilkan kinerja yang signifikan dan kemandirian bahasa [5]. Pernyataan mengenai kecepatan algoritma *SymSpell* masih menimbulkan keraguan, dimana ada yang menganggap hasil percobaan yang dilakukan sebelumnya masih belum menerapkan algoritma yang digunakan sebagai perbandingan dengan *SymSpell* secara optimal [8].

Dalam penelitian ini akan dilakukan penerapan dan analisa kinerja dari dua metode *spelling correction* yaitu metode *Burkhard Keller Tree* dan *SymSpell*. Dimana sebelumnya telah dinyatakan bahwa kecepatan merupakan faktor penting dalam *spell correction*, kedua metode dipilih untuk membuktikan pernyataan mengenai kecepatan dari metode *SymSpell* dengan membandingkannya dengan metode yang populer digunakan dalam *spell correction*, yaitu *BK Tree*. Selain itu, pemilihan metode *BK Tree* dipilih untuk membuktikan sebuah kesimpulan dari penelitian yang dilakukan sebelumnya, dimana menurut penelitian tersebut *BK Tree* memiliki tingkat presisi yang rendah ketika diterapkan dalam sistem *spell correction* bahasa Indonesia [9].

A. BK Tree

Burkhard Keller Tree atau *BK-Tree* adalah struktur data berupa *tree* yang dikembangkan oleh Walter Austin Burkhard dan Robert M. Keller pada tahun 1973 [9]. Struktur data *BK Tree* disesuaikan untuk ruang metriks diskrit, struktur data ini dapat digunakan untuk *spell correction* dengan memanfaatkan nilai *edit distance*. Penerapan *BK Tree* dapat dilihat pada Gbr. 1.



Gbr. 1 Contoh hasil penerapan BK Tree

Relasi dalam *BK Tree* disusun dengan ketentuan node tidak boleh memiliki child dengan nilai edit distance yang sama. Misal sebuah kata baru ditambahkan pada *root*, jika *root* memiliki *child* dengan nilai *edit distance* yang sama dengan kata baru tadi, kata baru tersebut akan dicoba ditambahkan menjadi *child* dari *node* dengan *edit distance* yang sama tadi, jika pada node ini juga ditemukan child dengan *edit distance* yang sama lagi, maka dilakukan hal yang sama, begitu seterusnya.

B. Symmetric Delete Spelling Correction

Symmetric Delete Spelling Correction atau *SymSpell* pertama kali dikembangkan oleh Wolf Garbe pada tahun 2012 [5]. Metode ini merupakan pengembangan dari penelitian metode *spell checker* milik Peter Norvig [10]. Pada metode milik Norvig kemungkinan kesalahan sebuah kata dibuat dengan semua operasi edit distance yaitu penyisipan (*insert*), penghapusan (*delete*), penggantian (*replace*), dan transposisi (*transpose*) dengan batas nilai *edit distance* yang ditentukan, kata-kata tersebut kemuadian dijadikan kamus untuk mengecek kesalahan ejaan kata. Sedangkan pada metode *SymSpell* hanya dilakukan proses penghapusan (*delete*) saja, hal ini mengurangi kompleksitas program dan dapat meningkatkan performanya secara signifikan [10]. Hasil kemungkinan kesalahan kata yang dihasilkan dengan metode *SymSpell* dapat dilihat pada Tabel I. Tabel tersebut menampilkan seluruh kemungkinan kesalahan pada kata "kamis" dengan metode SymSpell dengan nilai maksimal *edit distance* sebesar 2.

TABEL I
CONTOH HASIL PEMROSESAN KATA SYMSPELL

Kata	**Hasil**
kamis	kas, kamis, ais, kmis, kami, amis, kms, kmi, kams, ams, kis, kais, mis, ami, kai, kam

II. METODE PENELITIAN

Penelitian dilakukan dilakukan sesuai dengan langkah-langkah berikut.

A. Pengumpulan Data

Data yang digunakan dalam pembuatan sistem atau kamus adalah data dump basis data artikel Wikipedia Indonesia yang dapat diakses pada website Wikimedia[1] (organisasi pusat Wikipedia). Data yang didapat kemudian akan dilakukan proses normalisasi teks untuk menyiapkan teks agar lebih mudah dibaca dalam proses pembuatan kamus nantinya.

Sebagai uji coba, digunakan data yang didapat dari media sosial Twitter melalui Twitter API[2] yang disediakan oleh pihak Twitter untuk memberikan akses terprogram ke Twitter secara mendetail. Data yang diambil merupakan status pengguna atau dalam platform Twitter disebut tweets. Adapun untuk membatasi jumlah dan topik bahasan sampel, sejumlah lima puluh (50) tweets pengguna berbahasa Indonesia yang mengandung bahasan tentang "kampus merdeka" dipilih. Sama seperti data sebelumnya, data tweets ini kemudian juga akan dinormalisasi untuk membersihkan teks dari karakter yang tidak diperlukan supaya teks lebih relevan.

B. Pemrosesan Data

Pemrosesan data dilakukan pada kedua teks yang digunakan dalam pembuatan kamus dan pengujian. Dimana pertama untuk pemrosesan data teks corpus Wikipedia sebagai data untuk pembuatan kamus dilakukan dengan menghapus kata yang mengandung karakter alfabet non-latin, karena dalam Bahasa Indonesia hanya digunakan alfabet latin.

Untuk data pengujian yang sebelumnya diambil dari Twitter. Pemrosesan dilakukan dengan menormalisasi teks untuk membersihkan teks dari karakter yang tidak diperlukan dalam penelitian supaya teks lebih relevan. *Regular Expression* atau *RegEx* digunakan dalam proses normalisasi teks ini untuk memudahkan penulisan karena perlu dilakukan beberapa proses. *Regular Expression* merupakan sebuah teknik dalam bidang ilmu *Natural Language Processing* atau *Information Retrieval*, teknik ini mampu mendeskripsikan dan mengurai teks [11]. Regular Expression dapat menghapus atau menambahkan karakter dalam sebuah teks untuk menghasilkan teks sesuai kebutuhan.

Karakter atau kata yang dihapus dalam data pengujian adalah sebagai berikut, pertama penghapusan dilakukan pada *hastags*, *mentions*, dan *hyperlink*, karena hal-hal tersebut tidak terikat dengan penulisan ejaan kata, contohnya seperti mentions dimana hal tersebut merupakan *username* pengguna, sehingga hal-hal tersebut tidak diperlukan dalam uji coba penelitian ini. Kemudian tanda baca dan karakter selain alphanumerik dalam teks juga dihapus, karena sekali lagi karakter-karakter tersebut tidak akan terkait dengan penulisan ejaan kata yang diteliti dalam penelitian ini. Contoh hasil normalisasi teks ditunjukkan pada Tabel II.

TABEL II CONTOH HASIL NORMALISASI TEKS PENGUJIAN

Status	Teks
Asli	Program Merdeka Belajar Kampus Merdeka (MBKM) memberikan ruang bagi mahasiswa dalam mempelajari berbagai disiplin ilmu yang diminati. https://t.co/k1IugoxClC
Normalisasi	program merdeka belajar kampus merdeka mbkm memberikan ruang bagi mahasiswa dalam mempelajari berbagai disiplin ilmu yang diminati

C. Rancangan Sistem

Algoritma program dirancang terlebih dahulu sebelum diimplementasikan dalam Python[3], berikut rancangan algoritma program. Secara garis besar proses pada kedua algoritma *BK Tree* dan SymSpell dapat dibagi menjadi dua yaitu proses pembuatan kamus dan proses pencarian koreksi atau *lookup*.

Dalam algoritma *BK Tree* tahap-tahap penambahan kata pada proses pembuatan kamus dapat dilihat pada Gbr. 2.

```
Deklarasi
tree : node
kata : string
pointer : pointer
levenshtein : function

Algoritma
pointer = tree.root
DEF add(pointer, kata)
  IF pointer IS NULL
    tree.node APPEND kata
  ELSE
    dist = CALL levenshtein(kata, pointer.kata)
    IF dist IN pointer.children
      next = pointer.children WHERE dist IS SAME
      CALL add(next, kata)
    ELSE
      pointer.children APPEND kata
    ENDIF
  ENDIF
ENDDEF
```

Gbr. 2 Pseudocode penambahan kata proses pembuatan kamus metode BK Tree

Jika kata tidak ditemukan dalam kamus atau merupakan kata baru maka akan dilakukan dilakukan penelusuran pada *node tree* dimulai dari *node* teratas atau *root*. dihubungkan sebagai children untuk sebuah node jika node tidak memiliki nilai *edit distance* yang sama dengan edit distance dari kata pada *node* dan kata input. Sebaliknya jika *node* memiliki *child* dengan nilai *edit distance* yang sama dengan *edit distance* dari kata pada *node* dan kata input, maka penelusuran dilanjutkan pada *node child* itu, begitu seterusnya penelusuran dilakukan sampai sampai kata input menemukan *parent*-nya.

Selanjutnya algoritma proses pencarian koreksi metode *BK Tree* penelusuran branch pada tree dilakukan dengan beberapa ketentuan berdasarkan nilai edit distance, hal ini menyebabkan penelusuran kata hanya dilakukan pada branch yang memenuhi ketentuan tersebut, sehingga penelusuran dalam tree menjadi lebih cepat. Algoritma pencarian koreksi metode *BK Tree* ditunjukkan pada Gbr 3.

```
Deklarasi
tree : node
kata : string
max_edit : integer
pointer : pointer
results : array
levenshtein : function

Algoritma
pointer = tree.root
DEF find(pointer, kata, max_edit, results)
  dist = CALL levenshtein(kata, pointer.kata)
  IF dist <= max_edit
    results += kata
  ENDIF
  dmin = dist - max_edit
  dmax = dist + max_edit
  FOR child IN pointer.children
    IF dmin <= child.dist <= dmax
      CALL find(child, kata, max_edit, results)
    ENDIF
  ENDFOR
ENDDEF
```

Gbr. 3 Pseudocode proses pencarian koreksi metode BK Tree

Pada metode *SymSpell* secara garis besar proses juga dibagi menjadi dua, yaitu proses pembuatan kamus dan proses pencarian koreksi. Algoritma proses pembuatan kamus metode *SymSpell* ditunjukkan pada Gbr. 4. Setiap kata yang ditambahkan ke dalam kamus akan dibuatkan setiap kemungkinan kesalahan ejaan dengan menghapus beberapa karakter dengan batas *edit distance* yang ditentukan. Setiap kemungkinan kesalahan ejaan kemudian disimpan dalam kamus dengan kata aslinya sebagai daftar koreksi kata yang benar.

```
Deklarasi
deletes : dict
kata : string
max_edit : integer
edit_distance : integer
curr_distance : integer

Algoritma
curr_distance = 0
DEF edit(kata, edit_distance, curr_distance)
  edit_distance += 1
  FOR i IN range(curr_distance, len(kata))
    delete = kata[:i] + kata[i+1:]
    deletes[delete] APPEND kata
    IF edit_distance < max_edit
      CALL edit(kata, edit_distance, i)
    ENDIF
  ENDFOR
ENDDEF
```

Gbr. 4 Pseudocode penambahan kata proses pembuatan kamus metode SymSpell

Untuk proses pencarian koreksi metode *SymSpell* ditunjukkan dengan algoritma pada Gbr. 5.

[3] https://www.python.org/

```
Deklarasi
deletes : dict
kata : string
max_edit : integer
suggestion : array
levenshtein : function

Algoritma
INPUT kata, max_edit
IF kata IN deletes
  FOR koreksi IN deletes[kata]
    dist = CALL levenshtein(kata, koreksi)
    IF dist < max_edit
      suggestions APPEND koreksi
    ENDIF
  ENDFOR
ELSE
  FOR i IN range(len(kata))
    del = kata[:i] + kata[i+1:]
    IF del IN deletes
      FOR koreksi in deletes[del]
        dist = CALL levenshtein(kata, koreksi)
        IF dist <= max_edit
          suggestions APPEND koreksi
        ENDIF
      ENDFOR
    ENDIF
  ENDFOR
ENDIF
```

Gbr. 5 Pseudocode proses pencarian koreksi metode SymSpell

D. Pengujian Sistem

Pengujian dilakukan untuk sistem yang telah dibuat dengan mengimplementasikan kedua metode *BK Tree* dan SymSpell yang diteliti menggunakan perangkat dengan spesifikasi AMD A8-9600, RAM 8 GB, dan penyimpanan HDD. Sebagimana yang dijelaskan sebelumnya, data tweets yang telah dikumpulkan dan diproses sesuai kebutuhan akan digunakan sebagai data pengujian. Pengujian dilakukan untuk mengukur dan membandingkan kinerja kedua metode yang diterapkan dalam sistem *spell correction* untuk Bahasa Indonesia. Faktor-faktor yang diuji adalah kecepatan sistem, kebutuhan sistem, dan akurasi sistem.

1) Pengujian Kecepatan: Kecepatan merupakan faktor penting dalam sistem *spell correction* [4], sehingga pengujian ini merupakan salah satu langkah penting dalam penelitian ini. Pengujian kecepatan dilakukan untuk dua proses dalam sistem, yaitu proses pembuatan kamus dan proses pencarian koreksi. Pengujian akan dilakukan dengan input sampel tweets yang dijelaskan sebelumnya. Selain itu untuk mendapatkan hasil yang lebih mendalam, pengujian juga akan dilakukan untuk input berupa kata secara terpisah dengan beberapa skenario yaitu nilai maksimal edit distance yang berbeda dan jumlah kosakata dalam kamus yang berbeda. Pengujian dilakukan dengan mencatat waktu yang dibutuhkan sistem untuk menyelesaikan masing-masing proses.

2) Pengujian Kebutuhan Sistem: Pengujian kebutuhan sistem dilakukan terhadap memori yang dibutuhkan untuk menjalankan sistem pada masing-masing metode. Pengujian dilakukan dengan mencatat memori yang digunakan untuk memuat sistem serta memori yang digunakan untuk melakukan proses pencarian.

3) Pengujian Akurasi: Pengujian akurasi dilakukan dengan menganalisa hasil koreksi sampel *tweets*, perhitungan dilakukan terhadap jumlah sugesti koreksi kata yang benar dan koreksi kalimat yang benar untuk kedua metode *BK Tree* dan *SymSpell*. Data pengujian akan diambil dalam bentuk nilai *accuracy*, *precision*, dan *recall*. Untuk mendapatkan ketiga nilai tersebut digunakan perhitungan pada (1), (2), dan (3).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

$$Precision = \frac{TP}{TP + FP} \quad (2)$$

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

Evaluasi pengujian tersebut dapat divisualisasikan menggunakan *confusion* matriks pada Gbr. 6, dimana ditunjukkan benar atau salahnya hasil koreksi yang dihasilkan.

		Prediksi	
		Positif	**Negatif**
Asli	**Positif**	TP	FN
	Negatif	FP	TN

Gbr. 6 Ilustrasi confussion matriks untuk hasil pengujian

III. HASIL DAN PEMBAHASAN

A. Pengumpulan Data

Data yang digunakan dalam penelitian adalah data corpus artikel Wikipedia Indonesia dan data status Twitter atau *tweets*. Untuk corpus Wikipedia Indonesia yang nantinya digunakan sebagai data pembuatan kamus, data diambil pada website Wikimedia. Untuk data *tweets*, proses pengumpulan data dilakukan melalui Twitter API yang diakses menggunakan pustaka Tweepy[4] pada Python, dimana untuk mengakses data dilakukan dengan perintah pada Gbr. 7, dan salah satu contoh data *tweets* yang didapatkan dalam format JSON[5] ditunjukkan pada Gbr. 8.

```
import tweepy
from decouple import config

twitter = tweepy.Client(
    bearer_token=config('TWITTER_BEARER_TOKEN'),
    consumer_key=config('TWITTER_API_KEY'),
    consumer_secret=config('TWITTER_SECRET_KEY'),
    access_token=config('TWITTER_ACCESS_TOKEN'),
    access_token_secret=config('TWITTER_ACCESS_SECRET'))

query = '"kampus merdeka" lang:id -is:retweet'
tweets = twitter.search_recent_tweets(
    query=query,
    tweet_fields=['author_id', 'created_at'],
    sort_order='relevancy',
    max_results=50)
```

Gbr. 7 Kode program pengambilan data tweets

```
    "1592015302090719233": {
        "text": "Program Merdeka Belajar Kampus Merdeka (MBKM)
memberikan ruang bagi mahasiswa dalam mempelajari berbagai disiplin
ilmu yang diminati. https://t.co/k1IugoxClC",
        "author_id": 1544919581105754113,
        "created_at": "2022-11-14 04:43:20+00:00"
    },
```

Gbr. 8 Contoh data tweets yang diambil

B. *Pemrosesan Data*

Proses normalisasi teks dilakukan pada corpus Wikipedia Indonesia untuk pembuatan kamus dan teks tweets untuk pengujian. Seluruh teks pada corpus diformat menjadi lowercase atau menghilangkan penulisan huruf kapital, selain itu teks juga dibersihkan dari kata-kata dengan huruf atau aksara asing dengan perintah pada Gbr. 9.

```
def is_latin(text: str) -> bool:
    try:
        text.encode(encoding='utf-
8').decode('ascii')
    except UnicodeDecodeError:
        return False
    else:
        return True
```

Gbr. 9 Kode program pemrosesan corpus Wikipedia

Beberapa contoh kata dalam bahasa atau aksara asing yang ditemukan dalam corpus ditunjukkan pada Gbr 10. Dalam contoh tersebut ditemukan kata berbahasa Yunani serta Jerman, selain itu juga terdapat kata Bahasa Indonesia yang tidak dituliskan dengan ejaan yang benar.

```
['άνθρωπος',
 'keboedajaän'
 'mulaï',
 'soerabaïa',
 'linné',
 'grundzüge',
 'propädeutik'
 'zboże',
 'bhāṣā',
 'bāli']
```

Gbr. 10 Contoh kata asing yang dihapus dari corpus

Proses normalisasi teks berikutnya dilakukan pada kalimat tweets yang digunakan dalam proses pengujian. Proses normalisasi dilakukan menggunakan *regular expression* untuk menghilangkan karakter serta kata yang tidak diperlukan dalam penelitian, yaitu *hashtags*, *mentions*, *hyperlink*, tanda baca, dan karakter selain alphanumerik. Perintah tersebut ditunjukkan dalam kode program pada Gbr. 11.

```
import re

def clean_tweet(tweet: str) -> str:
    res = tweet.lower()
    res = re.sub("'", "", res)
    res = re.sub('@[A-Za-z0-9_]+','', res)
    res = re.sub('#[A-Za-z0-9_]+','', res)
    res = re.sub('http\S+', '', res)
    res = re.sub('[()!?]', ' ', res)
    res = re.sub('\[.*?\]',' ', res)
    res = re.sub('[^a-z0-9]',' ', res)
    res = re.sub(' +', ' ', res)
    return res
```

Gbr. 11 Kode program pemrosesan kalimat tweets

C. *Penerapan Sistem*

Sistem diterapkan berdasarkan rancangan algoritma yang telah dibuat sebelumnya. Pertama untuk sistem dengan metode *BK Tree* dapat dibagi menjadi dua proses, yaitu proses pembuatan kamus dan proses pencarian koreksi. Pada proses pembuatan kamus dengan metode *BK Tree* tiap kata yang dimasukkan akan ditambahkan ke dalam kamus dengan perintah pada Gbr. 12.

```
def add(self, word: str):
    def add_child(root: str, word: str):
        dist = self._distance(root, word)
        if dist > 0:
            for child in self._nodes[root].children:
                if dist == child[1]:
                    add_child(child[0], word)
                    break
            else:
                self._nodes[root].children.append((word, dist))

    if word in self._nodes:
        self._nodes[word].frequency += 1
        return

    self._nodes[word] = Node(frequency=1, children=[])
    add_child(self._root, word)
```

Gbr. 12 Kode program penambahan kata kamus metode BK Tree

Perintah tersebut akan menambahkan kata ke dalam kamus yang berupa susunan struktur data *tree* dengan menghitung nilai *edit distance*-nya. Setelah diproses dengan perintah tersebut kata akan ditambahkan menjadi *node* dengan memiliki *child* yang mempunyai nilai *edit distance* dengannya yang berbeda-beda, seperti yang ditunjukkan dalam Gbr. 13. Proses pembuatan kamus ini menghasilkan *node* dengan jumlah 1.597.416, dimana jumlah ini merupakan jumlah kosakata unik yang didapat dari corpus.

```
Node(frequency=8334, children=[('afrikat', 5), ('ringkas', 6), ('merekja', 2),
('bankir', 7), ('terkira', 4), ('menerka', 3), ('mereeka', 1)])
```

Gbr. 13 Contoh node di dalam kamus BK Tree

Proses pencarian metode *BK Tree* dilakukan menggunakan perhitungan nilai *edit distance*, dimana penelusuran *nodes* hanya akan dilakukan pada jangkauan nilai *edit distance* tertentu seperti yang dapat dilihat pada Gbr. 14.

```
def lookup(
    self,
    word: str,
    threshold: int
) -> List[SuggestItem]:
    if len(word) <= threshold:
        return word
    if word in self._nodes:
        return word

    suggestions: List[SuggestItem] = []
    if self._nodes:
        self._finder(self._root, word, threshold, suggestions)
    if not suggestions:
        return suggestions
    suggestions.sort(key=lambda x: (x.distance, -x.count))
    return suggestions
```

```
def _finder(
    self,
    root: str,
    word: str,
    threshold: int,
    suggestions: list
) -> None:
    root_node = self._nodes.get(root)
    dist = self._distance(root, word)
    if dist <= threshold:
        suggestions.append(SuggestItem(root, dist, root_node.frequency))

    for child in root_node.children:
        if dist - threshold <= child[1] <= dist + threshold:
            self._finder(child[0], word, threshold, suggestions)
```

Gbr. 14 Kode program pencarian koreksi metode BK Tree

Pada metode *SymSpell* proses utama sistem juga dibagi menjadi dua, yaitu proses pembuatan kamus dan proses pencarian koreksi. Untuk proses pembuatan kamus dilakukan dengan perintah pada Gbr. 15.

```
def create_dictionary_entry(self, key: str, count: int) -> bool:
    if key in self._words:
        count_previous = self._words[key]
        self._words[key] = increment_count(count, count_previous)
        return False
    self._words[key] = count
    edits = self._edits_prefix(key)
    for delete in edits:
        self._deletes[delete].append(key)
    return True

def _edits_prefix(self, key: str) -> Set[str]:
    def _edits(
        word: str,
        edit_distance: int,
        delete_words: Set[str],
        current_distance: int = 0,
    ) -> Set[str]:
        edit_distance += 1
        if not word:
            return delete_words
        for i in range(current_distance, len(word)):
            delete = word[:i] + word[i + 1 :]
            if delete in delete_words:
                continue
            delete_words.add(delete)
            if edit_distance < self._max_dictionary_edit_distance:
                self._edits(delete, edit_distance, delete_words, current_distance=i)
        return delete_words
```

Gbr. 15 Kode program program pembuatan kamus metode SymSpell

Untuk proses pencarian koreksi metode *SymSpell* dilakukan dengan perintah yang ditunjukkan pada Gbr. 16.

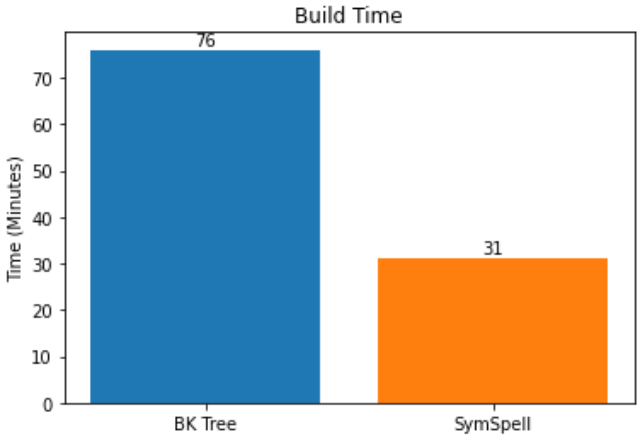
```
def lookup(
    self,
    phrase: str,
    verbosity: Verbosity,
    max_edit_distance: Optional[int] = None,
) -> List[SuggestItem]:
    suggestions: List[SuggestItem] = []
    phrase_len = len(phrase)

    considered_deletes = set()
    candidate_pointer = 0
    candidates = []
    candidates.append(phrase)
    while candidate_pointer < len(candidates):
        candidate = candidates[candidate_pointer]
        candidate_pointer += 1
        candidate_len = len(candidate)
        if candidate in self._deletes:
            dict_suggestions = self._deletes[candidate]
            for suggestion in dict_suggestions:
                if suggestion == phrase:
                    continue
                distance = distance_comparer(phrase, suggestion)
                if distance > max_edit_distance:
                    continue
                if distance <= max_edit_distance_2:
                    suggestion_count = self._words[suggestion]
                    item = SuggestItem(suggestion, distance, suggestion_count)
                    suggestions.append(item)
        for i in range(len(candidate)):
            delete = candidate[:i] + candidate[i + 1 :]
        if delete not in considered_deletes:
            considered_deletes.add(delete)
            candidates.append(delete)
    return suggestions
```

Gbr. 16 Kode program pencarian koreksi metode SymSpell

D. Pengujian Sistem

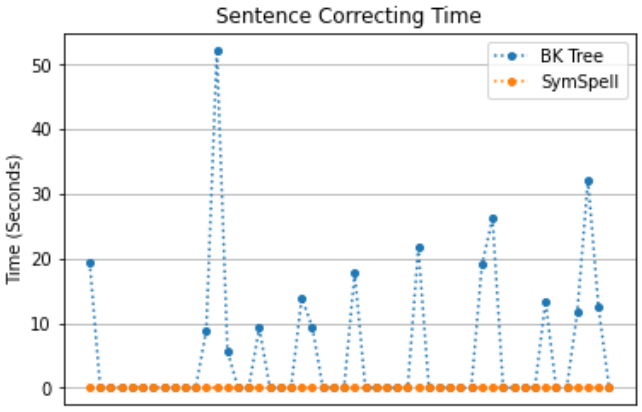
Proses pengujian dilakukan pada sistem yang telah dibuat sebelumnya dengan kedua metode *BK Tree* dan *SymSpell*, dimana dari proses pembuatan kamus pada kedua metode dihasilkan kamus dengan jumlah 1.597.416 kosakata, dengan tambahan pada metode *SymSpell* juga terdapat kamus *deletes* dimana berisi semua kemungkinan kesalahan kata yang dihasilkan sistem dengan jumlah 6.989.190 kata.

1) Pengujian Kecepatan: Pencatatan waktu dalam proses pembuatan kamus untuk masing-masing metode ditunjukkan pada Gbr. 17. Berdasarkan diagram tersebut metode *BK Tree* memiliki waktu yang lebih lama yakni 76 menit, dibandingkan metode *SymSpell* yang memiliki catatan waktu bernilai 27 menit.



Gbr. 17 Perbandingan catatan waktu proses pembuatan kamus

Pada proses pencarian koreksi yang diujikan denngan mengoreksi kalimat *tweets* yang telah dikumpulkan. Metode *SymSpell* juga memiliki pencatatan waktu yang jauh lebih cepat dibandingkan metode *BK Tree*, seperti yang ditunjukkan pada Gbr. 18. Pada metode *BK Tree* terdapat perbedaan waktu yang signifikan, dimana hal tersebut terjadi karena perbedaan jumlah kata yang perlu dikoreksi dalam kalimat dan perbedaan jumlah *node* yang perlu ditelusuri dalam *tree* untuk mencari sugesti kata.



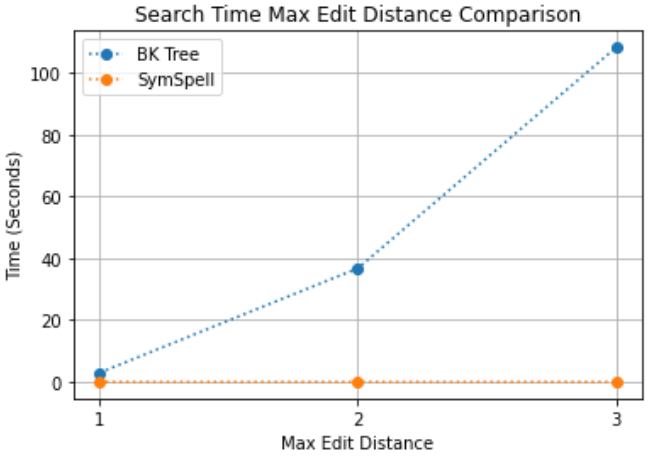
Gbr. 18 Perbandingan catatan waktu proses koreksi tweets

Lebih jelasnya, catatan waktu (detik) terbesar dan terkecil kedua metode ditunjukkan pada Tabel III. Sebagai catatan berdasarkan dokumentasi Python, akurasi pencatatan waktu juga dipengaruhi spesifikasi sistem, dalam pencatatan waktu kedua metode BK Tree dan SymSpell tercatat nilai nol (0) karena perubahan waktu lebih cepat dari akurasi waktu pada sistem yang digunakan.

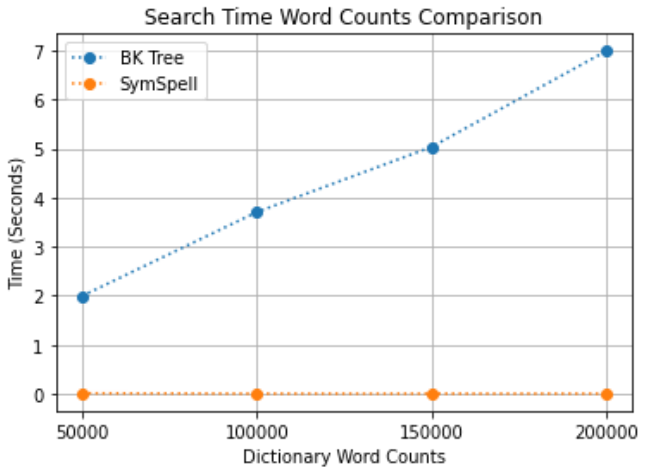
TABEL III
PERBANDINGAN CATATAN WAKTU MAKSIMAL DAN MINIMAL KOREKSI

Status	BK Tree	SymSpell
Max	52,24	0,05
Min	0,0	0,0

Selanjutnya pengujian dilakukan untuk menguji kecepatan metode dalam melakukan koreksi kata dalam skenario yang berbeda. Dalam pengujian ini digunakan satu kata dengan kesalahan ejaan yang diambil dari sampel tweets, yaitu kata "kripsi". Skenario pertama dilakukan dengan masukkan nilai *edit distance* maksimal yang berbeda, dimana hasilnya ditunjukkan pada Gbr. 19. Skenario kedua dilakukan menggunakan kamus dengan jumlah kata yang berbeda-beda di dalamnya dan digunakan nilai *edit distance* maksimal sebesar dua, dimana hasilnya ditunjukkan pada Gbr. 20.



Gbr. 19 Perbandingan catatan waktu koreksi pada masukkan edit distance maksimal yang berbeda

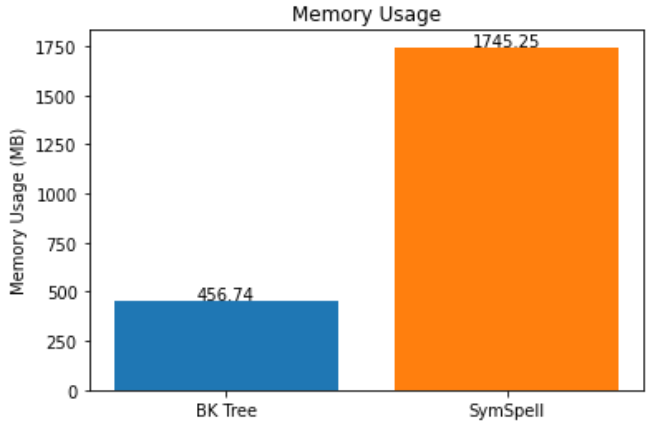


Gbr. 20 Perbandingan catatan waktu koreksi pada kamus dengan jumlah kata yang berbeda

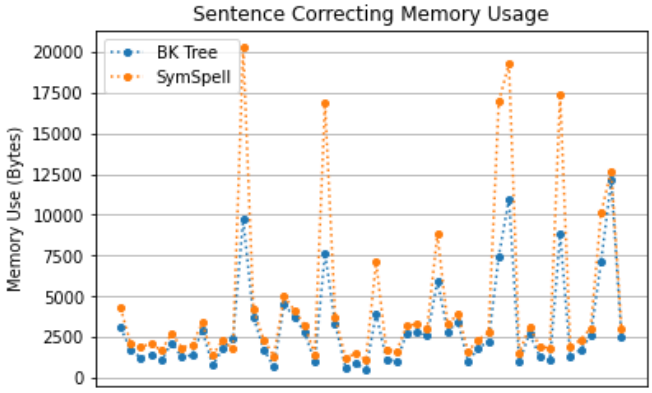
Pada pengujian tersebut terjadi peningkatan waktu pada metode *BK Tree*. Peningkatan waktu tersebut terjadi karena dengan meningkatnya nilai masukkan *edit distance* maksimal

dan jumlah kata pada kamus, maka semakin banyak *node* yang harus ditelusuri pada *tree*.

2) Pengujian Kebutuhan Sistem: Pengujian kebutuhan sistem dilakukan melalui pencatatan penggunaan memori pada sistem untuk memuat kamus serta untuk melakukan proses koreksi kata. Hasilnya ditunjukkan pada Gbr. 21 untuk penggunaan memori untuk memuat kamus dan pada Gbr. 22 untuk proses koreksi *tweets*.



Gbr. 21 Perbandingan penggunaan memori untuk memuat sistem

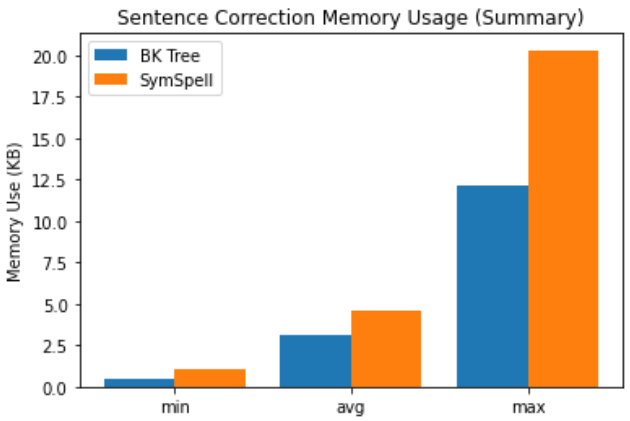


Gbr. 22 Perbandingan penggunaan memori proses koreksi tweets

Penggunaan memori pada metode *SymSpell* lebih tinggi dibandingkan metode *BK Tree* pada kedua kondisi. Rangkuman penggunaan memori pada proses koreksi juga ditunjukkan pada Tabel IV dan Gbr. 23.

TABEL IV
RANGKUMAN PENGGUNAAN MEMORI PROSES KOREKSI TWEETS

Data	Penggunaan Memori (KB)	
	BK Tree	SymSpell
Min	0,47	1,10
Average	3,09	4,64
Max	12,12	20,32



Gbr. 23 Grafik rangkuman perbandingan penggunaan memori proses koreksi tweets

3) Pengujian Akurasi: Uji coba akurasi dilakukan pada hasil koreksi kalimat tweets, dimana akan dihitung nilai *accuracy, precision,* dan *recall* dengan skala 0 sampai 1. Hasil koreksi dianggap benar jika sugesti yang diberikan mengandung kata yang tepat. Pengujian ditunjukkan pada Tabel V.

TABEL V
PERBANDINGAN HASIL PENGUJIAN AKURASI

No	BK Tree			SymSpell		
	Acc	Pre	Rec	Acc	Pre	Rec
1	0,69	1	0	0,69	1	0
2	1	1	1	1	1	1
3	1	1	1	1	1	1
4	0,88	1	0	0,88	1	0
5	1	1	1	1	1	1
6	1	1	1	1	1	1
7	1	1	1	1	1	1
8	1	1	1	1	1	1
9	1	1	1	1	1	1
10	1	1	1	1	1	1
11	1	1	1	1	1	1
12	1	1	1	1	1	1
13	0,79	0,43	0,33	0,79	0,43	0,33
14	0,85	1	0,33	0,85	1	0,33
15	1	1	1	1	1	1
16	1	1	1	1	1	1
17	0,98	1	0	0,98	1	0
18	1	1	1	1	1	1
19	1	1	1	1	1	1
20	1	1	1	1	1	1
21	0,95	0,5	1	0,95	0,5	1
22	1	1	1	1	1	1
23	1	1	1	1	1	1
24	1	1	1	1	1	1
25	1	1	1	1	1	1
26	0,93	0,5	1	0,93	0,5	1
27	1	1	1	1	1	1
28	1	1	1	1	1	1
29	1	1	1	1	1	1
30	1	1	1	1	1	1
31	1	1	1	1	1	1
32	0,94	1	0	0,94	1	0

No	BK Tree			SymSpell		
	Acc	Pre	Rec	Acc	Pre	Rec
33	1	1	1	1	1	1
34	1	1	1	1	1	1
35	1	1	1	1	1	1
36	1	1	1	1	1	1
37	0,88	1	0	0,88	1	0
38	0,65	0	0	0,65	0	0
39	0,64	0	0	0,64	0	0
40	1	1	1	1	1	1
41	1	1	1	1	1	1
42	0,73	1	0	0,73	1	0
43	1	1	1	1	1	1
44	1	1	1	1	1	1
45	0,93	1	0	0,93	1	0
46	1	1	1	1	1	1
47	0,67	1	0	0,67	1	0
48	0,86	0	0	0,86	0	0
49	0,9	0	0	0,9	0	0
50	1	1	1	1	1	1

Hasil pengujian menunjukkan kedua metode memiliki tingkat *accuracy, precision,* dan *recall* yang sama, karena kedua metode memberikan hasil koreksi yang sama.

IV. KESIMPULAN

Berdasarkan penelitian yang telah dilakukan terhadap perbandingan kinerja metode *BK Tree* dan *SymSpell* dalam sistem spell correction Bahasa Indonesia. Metode *SymSpell* terbukti memiliki keunggulan dalam faktor kecepatan, dimana metode ini unggul dalam proses pembuatan kamus dengan catatan waktu 31 menit dibandingkan metode *BK Tree* dengan catatan waktu 76 menit, dan juga pada proses pencarian koreksi dengan catatan waktu terbesar pada 0,05 detik dibandingkan metode *BK Tree* dengan catatan waktu terbesar pada 52,24 detik. Dalam faktor kebutuhan sistem metode *BK Tree* lebih unggul dengan penggunaan memori yang lebih sedikit dengan ukuran sebesar 456,74 MB dibandingkan metode *SymSpell* dengan penggunaan memori 1745,25 MB. Dalam faktor akurasi keduanya memiliki hasil yang sama, peneliti menemukan bahwa kedua metode tidak berpengaruh terhadap hasil sugesti. Dari penelitian juga didapatkan bahwa kualitas teks corpus yang digunakan untuk membuat kamus merupakan faktor penting dalam *spell correction*, dalam percobaan masih ditemukan *miss* koreksi karena corpus masih memiliki kosakata dengan kesalahan penulisan sehingga kata yang seharusnya salah dianggap benar karena terdapat dalam kamus.

V. SARAN

Berdasarkan penelitian yang telah dilakukan dapat dihasilkan saran yaitu, dapat digunakan corpus yang lebih berkualitas untuk digunakan dalam proses pembuatan kamus untuk melihat perbedaan hasil koreksi, dan digunakan metode-metode lain untuk mengetahui perbandingan kinerjanya atau untuk meningkatkan kinerja sistem *spell correction*, seperti metode untuk menentukan sugesti kata paling tepat berdasarkan konteks kata dalam kalimat.

UCAPAN TERIMA KASIH

Rasa terima kasih penulis sampaikan kepada Allah SWT atas berkat dan nikmat yang diberikan sehingga penelitian ini dapat diselesaikan dengan baik. Kemudian, kedua orang tua penulis atas doa dan dukungannya, serta bapak I Kadek Dwi Nuryana, S.T., M.Kom. selaku dosen pembimbing atas arahan dan bimbingannya. Rasa terima kasih tak lupa penulis ucapkan juga pada teman-teman Jurusan Teknik Informatika angkatan 2018 atas bantuan dan dukungannya.

REFERENSI

[1] Mohammed Nejja dan Abdellah Yousfi. 2018. “The Vocabulary and The Morphology in Spell Checker”. Procedia Computer Science 127: 76-81.

[2] Rakesh Kumar, Minu Bala, dan Kumar Sourabh. 2018. “A study of spell checking techniques for Indian Languages”. JK Research Journal in Mathematics and Computer Sciences, Vol.1, No.1.

[3] Jai Gupta, Zhen Qin, Michael Bendersky, dan Donald Metzler. 2019. “Personalized Online Spell Correction for Personal Search”. WWW ’19: The World Wide Web Conference: 2785–2791.

[4] G. Andrade., F. Teixeira, C. R. Xavier, R. S. Oliveira, L. C. Rocha, dan A. G. Evsukoff. 2012. “HASCH: High Performance Automatic Spell Checker for Portuguese Texts from The Web”. Procedia Computer Science 9: 403 – 411.

[5] Wolf Garbe. 2012. “1000x Faster Spelling Correction Algorithm”. seekstorm.com/blog/1000x-spelling-correction/, tanggal akses: Mei 2022.

[6] M. A. Budiman dan D. Rachmawati. 2021. “Implementation of Not So Naive and Burkhard-Keller Tree algorithms in Indonesian-Hokkien dictionary application”. Journal of Physics: Conference Series (2021).

[7] Chunchun Zhao dan Sartaj Sahni. 2019. “String correction using the DamerauLevenshtein distance”. BMC Bioinformatics 2019: 20.

[8] Wolf Garbe. 2017. “SymSpell vs. BK-Tree: 100x Faster Fuzzy String Search & Spell Checking”. https://seekstorm.com/blog/symspell-vs-bk-tree/, tanggal akses: Mei 2022.

[9] Mutamminah, Herry Sujaini, dan Rudy Dwi Nyoto. 2017. “Analisis Perbandingan Metode Spelling Corrector Peter Norvig dan Spelling Checker BK-Trees pada Kata Berbahasa Indonesia”. Jurnal Sistem dan Teknologi Informasi (JUSTIN) Vol.5, No.1 (2017).

[10] Rangga Gusdiwangsa. 2019. “Perbaikan Kesalahan Ejaan Dengan Metode Symspell Pada Kasus Tanya Jawab Dalam Bahasa Indonesia”. Universitas Komputer Indonesia.

[11] J. E. Friedl., 2006. *Mastering Regular Expressions, Third Edition*. Sebastopol: O'Reilly Media, Inc..