Mohamad Arvin Fadriansyah
2006596996

Tabel Perbandingan Antar Algoritma

	Levensthein			Damerau-Levenshtein		
	Run Time	**Best Match Accuracy**	**Candidate Accuracy**	**Run Time**	**Best Match Accuracy**	**Candidate Accuracy**
Trie	8.979200839996338 seconds	0.44	1.0	12.735947608947754 seconds	0.48	1.0
Dict	17.467824935913086 Seconds	0.44	1.0	151.0642204284668 seconds	0.48	1.0

Nomor a

Penjelasan Kelas Performance

Kelas Peforma adalah kelas yang berguna untuk Menghitung peforma tiap model dengan menggunakan metriks best match accuracy dan candidate accuracy

1. **Atributnya**

```
model = Jenis algoritma yang digunakan untuk pencarian kandidat kata yang benar
typo_sample_dataset = list 2 dimensi yang tiap row nya mengandung typo sample
keys dan list kemungkinan non words error nya
```

2. **Inisialisasi**

```
def __init__(self, model, typo_sample_dataset):
    self.__model = model
    self.__typo_sample_dataset = typo_sample_dataset
```

3. **Function calculate_performance**

Penjelasan:

```
    """
    Memanggil semua jenis fungsi perhitungan akurasinya dan menghitung runtimenya.
    runtime mulai dihitung ketika kode yang berguna untuk mencari candidate
    words dieksekusi, dan runtime berhenti ketika baris code tersebut
    telah selesai dieksekusi
    """
```

- Perhitungan waktu dimulai

```
start_time = time.time()  # Waktu mulai
```

Mohamad Arvin Fadriansyah
2006596996

- Menghitung peforma dari tiap algoritma dengan best_match_accuracy dan candidate_accuracy

```
        # Best match accuracy
        best_match_accuracy = self.best_match_accuracy()

        # Candidate accuracy
        candidate_accuracy = self.candidate_accuracy()
```

- Setelah perhitungan selesai maka kita langsung menyimpan waktu berakhirnya. Dan kita menghitung durasinya.

```
    end_time = time.time()  # Waktu berakhir

    duration = end_time - start_time  # Selisih waktu mulai dan berkahirnya
```

- Return value nya adalah sebagai berikut:

```
        return {
            'cand_acc': candidate_accuracy,
            'best_acc': best_match_accuracy,
            'time': duration
        }
```

4. **Penjelasan function candidate_accuracy**
   - Penjelasannya berada pada snippet code seperti sebagai berikut,
   - Variable max_cost nya sebesar **2** (self.__mode.get_candidates(typo, **2**))

Mohamad Arvin Fadriansyah
2006596996

```python
def candidate_accuracy(self) -> float:
    """
    Menghitung candidate_accuracy dengan rumus m/n
    m = Jumlah instance dari correct word yang ada pada list of candidates.
    n = Total non-word errors yang ada pada test dataset.
    """
    m = 0
    n = 0
    for key, typo_list in self.__typo_sample_dataset:
        for typo in typo_list:

            # kata non-word errors kita cari kemungkinan kandidat kata yang benarnya
            candidates = self.__model.get_candidates(typo, 2)
            n += 1

            for candidate in candidates:

                # Jika kata yang benar ditemukan pada kemungkinan kandidatnya
                if (key == candidate):
                    m += 1

    return m / n
```

5. **Penjelasan function best_match_accuracy**
   - Penjelasannya berada pada snippet code seperti sebagai berikut,
   - Variable max_cost nya sebesar **2** (self.__mode.get_candidates(typo, **2**))

Mohamad Arvin Fadriansyah
2006596996

```python
def best_match_accuracy(self) -> float:
    """
    Menghitung best_match_accuracy dengan rumus m/n
    m = Jumlah instance dari correct word yang ada pada list of candidates
    dan berada di posisi pertama pada list.
    n = Total non-word errors yang ada pada test dataset.
    """
    m = 0
    n = 0
    for key, typo_list in self.__typo_sample_dataset:
        for typo in typo_list:

            # kata non-word errors kita cari kemungkinan kandidat kata yang benarnya
            candidates = self.__model.get_candidates(typo, 2)
            n += 1

            # Jika kata yang benar ditemukan pada pada index pertama candidatesnya
            if (key == candidates[0]):
                m += 1

    return m / n
```

Penjelasan kelas Main

1. **Mengubah dictionary typo_sample_dataset menjadi list dua dimensi agar lebih readable dan bersesuaian dengan kebutuhan untuk perhitungan metriks akurasi**

```python
"""
Ubah Data dari awalnya dictionary menjadi bentuk list 2 dimensi
yang tiap row nya mengandung typo sample keys dan list kemungkinan non words error nya
"""
typo_sample_dataset_modified = []
for key, typo_list in typo_sample_dataset.items():
    typos=[]
    for typo_data in typo_list:
        typos.append(typo_data['typo'])
    typo_sample_dataset_modified.append([key,typos])
typo_sample_dataset = typo_sample_dataset_modified
print(typo_sample_dataset)
```

```
[6]  ✓ 0.0s                                                                    Python
[['komposisi', ['ikmposisi', 'omposisi', 'uiomposisi', 'ioposisi', 'mposisi', 'uomposisi', 'iomposisi', 'pmposisi', 'oimposisi', 'pkmposisi', 'ikomposi
```

2. **Inisialisasi objek yang argumennya adalah algoritma pencarian kandidat kata non words error dan dataset typo_sample_dataset yang isinya 5 buah kata random yang terdiri dari kata benar dan kemungkinan typonya. Kemudian hitung masing-masing peforma dari tiap algoritma**

```python
# Dictionary untuk formatting
performances = {"lev_trie": performance_of_lev_trie.calculate_performance(),
                "dalev_trie": performance_of_dalev_trie.calculate_performance(),
                "lev_dict": performance_of_lev_dict.calculate_performance(),
                "dalev_dict" :performance_of_dalev_dict.calculate_performance()
                }
```

Mohamad Arvin Fadriansyah
2006596996

3. **Berikut adalah rangkuman hasilnya:**

```
1  print(format_report(nama, npm,performances))
✓ 0.0s

Mohamad Arvin Fadriansyah - 2006596996
---------- lev_trie ----------
Best Accuracy: 0.44
Candidate Accuracy: 1.0
Time: 8.979200839996338 seconds
---------- dalev_trie ----------
Best Accuracy: 0.48
Candidate Accuracy: 1.0
Time: 12.735947608947754 seconds
---------- lev_dict ----------
Best Accuracy: 0.44
Candidate Accuracy: 1.0
Time: 17.467824935913086 seconds
---------- dalev_dict ----------
Best Accuracy: 0.48
Candidate Accuracy: 1.0
Time: 151.0642204284668 seconds
```

<u>Nomor b</u>

Dari Segi kecepatan struktur data Trie lebih cepat jika dibandingakan dengan dictionary baik pada algoritama Levenshtein maupun Damerau-Levenshtein
- Pada algoritma Levenshtein, Trie jauh lebih cepat (8.98 detik) dibandingkan Dictionary (17.47 detik).

- Pada algoritma Damerau-Levenshtein, Trie (12.74 detik) juga jauh lebih cepat dibandingkan Dictionary (151.06 detik).

Hal ini bisa terjadi dikarenakan Trie memiliki efisiensi dalam operasi pencarian string karena representasi treenya memungkinkan pencocokan lebih cepat. Di sisi lain, Dictionary, yang menggunakan hash table atau mapping, lebih lambat dalam operasi yang melibatkan perbandingan string satu per satu.

Dari sisi akurasi, Algoritma Damerau-Levenshtein lebih akurat (Best Accuracy 0.48) dibandingkan dengan Levenshtein (Best Accuracy 0.44). Algoritma Damerau-Levenshtein memperhitungkan transposisi karakter, yang membuatnya lebih efektif dalam menangani kesalahan ketik atau pertukaran karakter, sehingga menghasilkan prediksi yang lebih baik.

Mohamad Arvin Fadriansyah
2006596996

Struktur data tidak mempengaruhi akurasi karena baik Trie maupun Dictionary memberikan akurasi yang sama. Hanya Algoritma yang mempengaruhi perbedaan akurasi.

Nomor c

Tidak, Struktur data tidak mempengaruhi akurasi secara langsung karena pada percobaan yang kita lakukan baik Trie maupun Dictionary menghasilkan Best Accuracy yang sama (misalnya, lev_trie dan lev_dict sama-sama memiliki Best Accuracy 0.44). Akurasi lebih dipengaruhi dari algoritma yang digunakan, (Levenshtein atau Damerau-Levenshtein)

Hal ini bisa terjadi dikarenakan struktur data hanya digunakan untuk cara penyimpanan dan pencarian data. Trie dan dictionary hanyalah alat untuk mengakses data dengan cara yang berbeda. Struktur data hanya berpengaruh pada waktu komputasi.