Asdos PJ: Alvin Xavier Rakha W.
CSCE604231: PENGOLAHAN BAHASA MANUSIA (NLP)
September 18, 2024

Tugas Lab 3 (TL 3)
Analsis Perbandingan Algoritma dan Struktur Data untuk *Spelling Correction*
***Deadline*: Jumat 20 September 2024 jam 22:00**

Tugas ini meminta Anda untuk menganalisis akurasi dan *runtime* dari dua algoritma *edit distance*, yaitu *Levenshtein Distance* dan *Damerau-Levenshtein Distance*. Kedua algoritma ini akan diuji menggunakan dua struktur data yang berbeda, yaitu *Trie* dan *Dictionary*.

1. Apa yang Perlu Dipahami?

Untuk menyelesaikan tugas ini, Anda perlu memahami struktur data *Dictionary* dan *Trie*, algoritma Damerau–Levenshtein dan Levenshtein Distance, serta cara menghitung akurasinya.

1.1. **Struktur Data *Trie*.** *Trie* adalah sebuah struktur data yang berbentuk seperti pohon yang biasa digunakan untuk menyimpan himpunan kata yang setiap hurufnya disimpan dalam sebuah node yang telah tersusun dengan root node sebagai prefix dari suatu kata dan child nodes yang berisi huruf-huruf dari suatu kata-kata yang memiliki prefix yang sama.

1.2. **Struktur Data *Dictionary*.** *Dictionary* adalah struktur data yang memetakan kunci (*key*) ke nilai (*value*). Di Python, dictionary diimplementasikan sebagai hash map. *Key* bersifat unik, dan setiap *key* terhubung ke satu nilai melalui *hashing*. Oleh karena itu, operasi pencarian, penambahan, dan penghapusan dalam dictionary umumnya sangat cepat.

1.3. **Levenshtein Distance.** *Levenshtein Distance* adalah metode untuk mengukur jarak antara *source string* dan *target string* [1]. Jarak yang dimaksud adalah jumlah minimum edit yang dibutuhkan untuk mengubah dari *source string* ke *target string*. Pada *Levenshtein Distance*, terdapat beberapa *edit operation* yang bisa dilakukan, yaitu *insertion*, *substitution*, dan *deletion*. Untuk setiap aksi edit, akan dicari *cost* yang paling minimal antara ketiga jenis aksi tersebut.

1.4. **Damerau-Levenshtein Distance.** *Damerau-Levenshtein Distance* kurang lebih sama seperti *Levenshtein Distance*, tetapi terdapat satu tambahan aksi untuk melakukan edit terhadap *string*, yaitu *transposition*. *Transpostion* dilakukan pada dua buah kata yang bersebelahan pada *source string* dan jika dua buah kata tersebut ditukar posisinya, maka *source string* akan similar dengan *target string*.

1.5. **Menghitung Akurasi.** Pada kasus *spelling correction* ini, metode untuk menghitung akurasi yang akan digunakan terdapat pada *paper* [2]. Berdasarkan paper tersebut terdapat dua jenis akurasi, yaitu *Candidate Accuracy* dan *Best Match Accuracy*:

$$Candidate\_Accuracy = \frac{M}{N} * 100$$

1) M = Jumlah *instance* dari *correct word* yang ada pada *list of candidates*.
2) N = Total non-word errors yang ada pada *test dataset*.

$$Best\_Match\_Accuracy = \frac{M}{N} * 100$$

1) M = Jumlah *instance* dari *correct word* yang ada pada *list of candidates* dan berada di posisi pertama pada *list*.
2) N = Total *non-word errors* yang ada pada *test dataset*.

2. Apa yang Harus Dikerjakan?

Setelah memahami informasi di atas, berikut adalah beberapa hal yang perlu dilakukan.

1) *Clone*/unduh repositori GitHub ini yang berisi:
   - Algoritma edit distance yang masing-masing terdapat pada folder trie_structure dan dict_structure (**algoritma-algoritma tersebut sudah siap pakai, jadi tidak ada yang perlu diubah pada code tersebut.**)
   - `main.ipynb`: Kode untuk menguji algoritma *edit distance* yang perlu dilengkapi.
   - `performance.py`: Kode untuk menghitung akurasi yang perlu dilengkapi.
   - `utils.py`: Beberapa *function* yang dapat membantu pengerjaan.

2) *Install dependency* yang diperlukan sesuai `requirements.txt`

3) Unduh sebuah file non-word error dataset untuk Bahasa Indonesia bernama SALTIK, yang dibuat oleh [2].

4) Download satu buah file .txt yang berisikan dataset KBBI pada Scele. ***Warning*: Dataset KBBI memiliki Hak Cipta yang mengakibatkan tidak boleh disebarluaskan sehingga para mahasiswa harap diharapkan tidak menyebarkan *code*/dataset ini**.

5) Atur parameter max_cost=2 pada saat memanggil *function* untuk mendapatkan kandidat kata.

6) Pada `main.ipynb`, isi nama dan NPM Anda pada variabel `npm`. Perhatikan bahwa NPM Anda akan memengaruhi sampel dari dataset *testing* yang Anda dapatkan.

7) Implementasikan algoritma perhitungan akurasi dan *runtime* pada `performance.py` (**Note: *Runtime* mulai dihitung ketika kode yang berguna untuk mencari *candidate words* dieksekusi, dan *runtime* berhenti ketika baris code tersebut telah selesai dieksekusi**)

8) Gunakan implementasi tersebut untuk menguji keempat algoritma *edit distance* pada `main.ipynb`.

9) Buat format *output* dari hasil perhitungan performa sebagaimana pada fungsi `format_report()` yang sudah tersedia di `utils.py` (boleh diubah sesuai kebutuhan). **Note: luaran hasil performa tidak perlu dibulatkan dan hasil performa di bawah ini hanya contoh, bukan hasil yang sebenarnya. Jangan lupa print nama dan NPM Anda bersamaan dengan luaran tersebut.**

```
[ nana - npm ]
---------- lev_trie ----------
best accuracy : 0.05
candidate accuracy : 1.0
total_time : 0.6771059036254883
---------- dalev_trie ----------
best accuracy : 0.05
candidate accuracy : 1.0
total_time : 0.6639831066131592
---------- lev_dict ----------
best accuracy : 0.05
candidate accuracy : 1.0
total_time : 0.89540696144104
---------- dalev_dict ----------
best accuracy : 0.05
candidate accuracy : 1.0
total_time : 6.538262605667114
```

10) Buat tabel perbandingan antar algoritma dengan format seperti Table 1 dan lampirkan ke dalam laporan. Bulatkan value ke tiga angka dibelakang koma.

	Levenshtein			**Damerau-Levenshtein**		
	Run Time	**Best Match Accuracy**	**Candidate Accuracy**	**Run Time**	**Best Match Accuracy**	**Candidate Accuracy**
Trie						
Dict						

TABLE 1. *Template* Tabel untuk Perbandingan antar Algoritma

11) Buat laporan dengan format `pdf` yang berisikan:
   (a) Penjelasan singkat mengenai algoritma perhitungan akurasi
   (b) Perbandingan antar struktur data dan algoritma, mana yang lebih cepat, mana yang lebih akurat, mengapa demikian?
   (c) Apakah struktur data mempengaruhi akurasi? Jika iya, mana yang lebih akurat? Jelaskan mengapa hal itu bisa terjadi!

3. Pengumpulan Tugas

File yang perlu dikumpulkan (satukan dalam berkas *zip*):

1) *File* laporan dengan aturan nama `NLP-TL3-[NamaAnda].pdf`.
   Contoh: `NLP-TL3-AlvinXavierRakhaWardhana.pdf`.
2) *File* `main.ipynb` **dan** `performance.py` yang sudah dilengkapi.

Catatan: Pastikan *file* laporan tidak bermasalah **dan** program bisa dijalankan tanpa *error*. Akan ada pengurangan poin jika ada masalah terkait kedua hal tersebut.

References

[1] R. Haldar and D. Mukhopadhyay, "Levenshtein distance technique in dictionary lookup methods: An improved approach," *ArXiv*, vol. abs/1101.1232, 2011. [Online]. Available: https://api.semanticscholar.org/CorpusID:1049293

[2] H. Audah, A. Yuliawati, and I. Alfina, "A comparison between symspell and a combination of damerau-levenshtein distance with the trie data structure," 10 2023, pp. 1–6.