Penerapan Algoritma Knuth-Morris-Pratt dan Booyer-Moore pada Pengoreksian Kata Tidak Baku dalam Teks Bahasa Indonesia

Mohamad Daffa Argakoesoemah - 13520118
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: argakoesoemahmdaffa@gmail.com

***Abstrak*—Bahasa Indonesia memiliki banyak sekali kosakata. Di samping itu, terdapat perbedaan pengucapan ataupun penulisan bahasa Indonesia di berbagai daerah atau situasi. Hal tersebut menjadi salah satu penyebab munculnya kata baku dan tidak baku. Penggunaan kata baku dalam bahasa Indonesia menjadi sangat penting ketika bahasa Indonesia digunakan dalam konteks formal, seperti dalam penulisan dokumen resmi. Maka dari itu, pengoreksi otomatis penggunaan kata baku dalam teks bahasa Indonesia menjadi berguna. Pengoreksi otomatis ini dapat memanfaatkan algoritma pencocokan *string*. Algoritma pencocokan *string* dapat menemukan kecocokan kata tidak baku sebagai pola pada sebuah teks bahasa Indonesia. Selanjutnya, kata yang tidak baku tersebut dapat diganti menjadi kata baku. Algoritma pencocokan *string* yang dapat digunakan, antara lain adalah algoritma Knuth-Morris-Pratt dan Booyer-Moore.**

Kata kunci—kata baku; pencocokan string; teks bahasa Indonesia; koreksi

I. PENDAHULUAN

Bahasa Indonesia merupakan bahasa nasional dan resmi negara Indonesia yang dipakai sehari-hari oleh penduduknya di samping pemakaian bahasa daerah. Bahasa ini adalah bahasa yang kaya kosakata. Per tahun 2018, bahasa Indonesia memiliki sekitar 127 ribu kosakata. Kosakata ini akan terus bertambah seiring waktu karena usulan-usulan yang berdatangan dari berbagai bahasa daerah. Hal ini karena Indonesia juga memiliki sangat banyak bahasa daerah.

Berdasarkan paragraf sebelumnya, pengucapan dan penulisan bahasa Indonesia bisa berbeda-beda. Hal ini bisa bergantung konteks ataupun daerah. Oleh karena itu, banyak bermunculan bahasa informal yang dipakai dalam percakapan sehari-hari. Namun, bahasa informal tersebut tidak bisa dipakai dalam konteks formal. Kemampuan seseorang menggunakan bahasa Indonesia dalam konteks formal dinilai cukup penting, misalnya dalam pembuatan dokumen resmi. Penggunaan bahasa Indonesia dalam konteks formal harus mengikuti Panduan Umum Ejaan Bahasa Indonesia (PUEBI) dan Kamus Besar Bahasa Indonesia (KBBI). Salah satu kaidah yang wajib diikuti adalah penggunaan kata baku.

Pengoreksian teks berbahasa Indonesia yang mempunyai kata tidak baku dapat dilakukan dengan algoritma pencocokan *string*. Hal ini dapat mempermudah seseorang untuk melakukan koreksi pada teks miliknya, apalagi jika orang tersebut tidak memiliki pengetahuan yang cukup mengenai kata baku dan tidak baku dalam bahasa Indonesia. Algoritma pencocokan *string* yang dapat digunakan adalah algoritma Knuth-Morris-Pratt dan Booyer-Moore. Makalah ini juga akan memperlihatkan perbandingan kecepatan antara kedua algoritma tersebut pada pengoreksian kata tidak baku.

II. LANDASAN TEORI

A. Pencocokan String

1. Definisi Pencocokan *String*

Pencocokan *string* atau *string matching* adalah permasalahan untuk mencari kemunculan sebuah pola *string* pada sebuah teks *string*. Terdapat beberapa algoritma pencocokan *string*, antara lain *brute force*, Knuth-Morris-Pratt (KMP), Booyer-Moore, dan Rabin Karp. Pencocokan *string* juga bisa dilakukan menggunakan *regular expression*.

2. Algoritma Knuth-Morris-Pratt (KMP)

Algoritma Knuth-Morris-Pratt (KMP) adalah salah satu algoritma pencocokan *string* yang melakukan pencocokan antara pola dan teks dari kiri ke kanan. Pada algoritma ini, jika terjadi ketidakcocokan karakter antara teks T dan pola P pada P[j] (T[i] tidak sama dengan P[j]), pola akan digeser sebanyak ukuran prefiks pada P[0..j-1] yang juga merupakan sufiks pada [1..j-1]. Hal ini dilakukan agar tidak terjadi pencocokan yang tidak berguna. Ilustrasi algoritma KMP dapat dilihat pada Fig. 1.

Algoritma KMP adalah salah satu algoritma pencocokan *string* yang melakukan pencocokan antara pola dan teks dari kiri ke kanan. Pada algoritma ini, jika terjadi ketidakcocokan karakter antara teks T dan pola P pada P[j] (T[i] tidak sama dengan P[j]), pola akan digeser sebanyak ukuran prefiks pada P[0..j-1] yang juga merupakan sufiks pada [1..j-1]. Hal ini dilakukan agar tidak terjadi pencocokan yang tidak berguna. Ilustrasi algoritma KMP dapat dilihat pada Fig. 1.

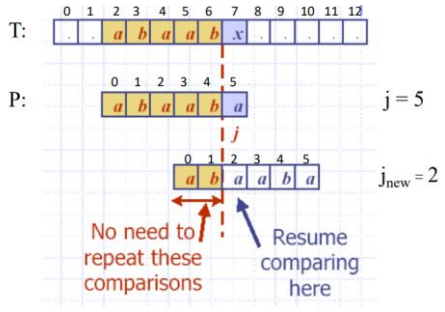


Fig. 1. Ilustrasi algoritma KMP
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-*string*-2021.pdf

Algoritma KMP melakukan *preprocessing* pola untuk mencari kesamaan prefiks pada pola dengan pola itu sendiri. Hal ini biasanya diimplementasikan dengan sebuah fungsi yang bernama *border function* atau fungsi pinggiran. *Border function* b(k) menghasilkan ukuran prefiks terbesar pada P[0..k] yang juga merupakan sufiks pada P[1..k].

Misalnya m adalah panjang pola dan n adalah panjang teks. Kompleksitas pada fungsi pinggiran adalah O(m), sedangkan kompleksitas pada pencocokan *string* adalah O(n). Maka dari itu, kompleksitas algoritma KMP adalah O(m+n).

3. Algoritma Booyer-Moore

Berbeda dengan algoritma KMP, algoritma Booyer-Moore melakukan pencocokan *string* dengan membaca pola dari kanan ke kiri. Algoritma ini didasarkan pada dua teknik, yaitu teknik *looking-glass* dan *character-jump*. Teknik *looking-glass* mencari pola pada teks dengan membaca pola dari kanan ke kiri mulai dari karakter terakhir atau secara terbalik seperti yang telah disebutkan sebelumnya. Teknik *character-jump* terdiri dari tiga kasus yang terjadi ketika ketidakcocokan karakter pada teks indeks ke-i yang mempunyai karakter x (T[i] = x) dan karakter pada pola P indeks ke-j (P[j]) tidak sama dengan T[i].

Kasus pertama terjadi ketika terdapat x pada P. Yang dilakukan adalah menggeser P ke kanan sedemikian sehingga dengan kemunculan terakhir dari x pada P sejajar dengan T[i]. Ilustrasi kasus pertama dapat dlihat pada Fig. 2.

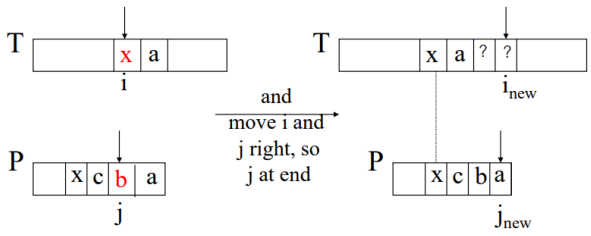


Fig. 2. Ilustasi kasus pertama teknik *character-jump*
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-*string*-2021.pdf

Kasus kedua terjadi ketika terdapat x pada P, tetapi pergeseran P ke kanan sehingga seperti kasus pertama tidak memungkinkan. Solusinya adalah menggseser P satu karakter ke kanan sehingga sejajar dengan T[i+1]. Ilustrasi kasus kedua dapat dlihat pada Fig. 3.

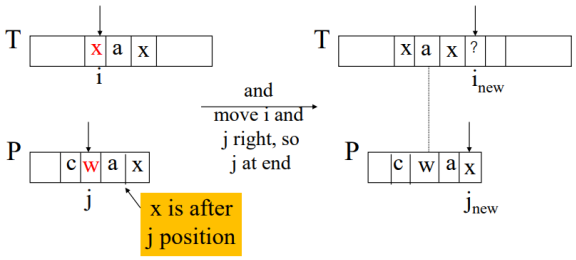


Fig. 3. Ilustasi kasus kedua teknik *character-jump*
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-*string*-2021.pdf

Kasus ketiga terjadi ketika kasus pertama dan kasus kedua tidak terjadi. Solusinya adalah menggeser P sehingga P[0] sejajar dengan T[i+1]. Ilustrasi kasus ketiga dapat dilihat pada Fig. 4.

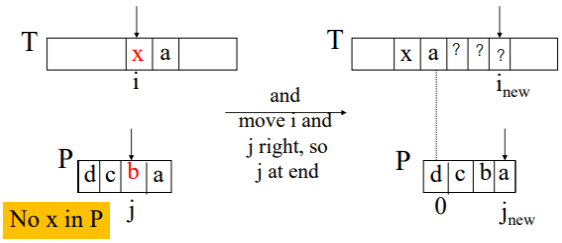


Fig. 4. Ilustasi kasus ketiga teknik *character-jump*
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-*string*-2021.pdf

Algoritma Booyer-Moore juga melakukan *preprocessing*, seperti algoritma KMP. Algoritma Booyer-Moore memproses pola dan alfabet yang ada pada teks untuk membuat fungsi bernama *last occurence*. Misalnya L() adalah fungsi *last occurrence*. L(x) mengembalikan indeks terbesar kemunculan x pada pola atau -1 jika indeks tidak ada kemunculan. Ilustrasi algoritma ini beserta tabel fungsi *last occurrence*-nya dapat dilihat pada Fig 5. Kompleksitas algoritma ini dalam scenario terburuk adalah O(nm+A) dengan n adalah panjang teks, m adalah panjang pola, dan A adalah banyak alfabet pada teks.

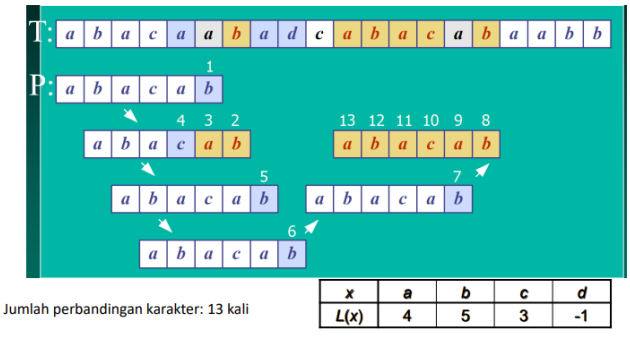


Fig. 5. Ilustasi algoritma Booyer-Moorre
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-*string*-2021.pdf

B. *Kata Baku dan Tidak Baku*

Kata baku merupakan kata yang sesuai dengan kaidah bahasa Indonesia. Sebaliknya, kata tidak baku merupakan kata yang tidak sesuai kaidah bahasa Indonesia. Kaidah bahasa Indonesia mengacu kepada Pedoman Umum Ejaan Bahasa Indonesia (PUEBI) dan Kamus Besar Bahasa Indonesia (KBBI). Kata baku biasa dipakai dalam situasi atau konteks resmi, seperti pada pidato kenegaraan, karya ilmiah, dll. Bahasa baku mempunyai ciri-ciri sebagai berikut:

- Tidak dipengaruhi bahasa daerah
- Tidak dipengaruhi bahasa asing
- Bukan merupakan ragam bahasa percakapan
- Pemakaian imbuhan secara eksplisit
- Pemakaian sesuai konteks kalimat
- Bukan merupakan kata rancu
- Tidak mengandung pleonasme

Di sisi lain, kata tidak baku biasa dipakai dalam situasi yang tidak formal, seperti percakapan sehari-hari dalam ruang lingkup keluarga. Kata-kata tidak baku dapat berupa beberapa hal berikut, yaitu:

- Kata dari dialek bahasa Indonesia yang ada
- Kata serapan bahasa daerah yang belum berterima
- Kata bahasa asing yang tidak memenuhi persyaratan ejaan bahasa Indonesia
- Kata bahasa Indonesia yang dieja sebagai bahasa asing
- Kata bentukan yang tidak mengikuti kaidah yang berlaku

III. Impelementasi Program Pengoreksi Kata Tidak Baku

Program pengoreksi bahasa tidak baku diimplementasikan menggunakan bahasa pemrograman Python. Di sisi lain, daftar kata baku dan tidak baku dalam bahasa Indonesia disimpan di dalam basis data relasional MariaDB. Daftar pasangan kata baku dan tidak baku diambil dari repositori Github pada link https://github.com/lantip/baku-tidak-baku. Skema basis data dapat dilihat pada Fig. 6.

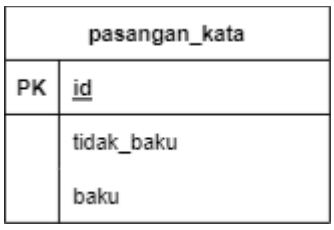


Fig. 6. Skema basis data kata baku
Sumber: Dokumentasi pribadi

Pada basis data ini, hanya terdapat satu tabel bernama pasangan_kata yang terdiri dari tiga atribut, yaitu atribut id sebagai *primary key*, atribut bernama tidak_baku yang berisi kata tidak baku, dan atribut bernama baku yang berisi kata baku. Tabel ini terdiri dari 1451 baris. Gambar potongan isi tabel ini dapat dilihat pada Fig. 7.

id	tidak_baku	baku
1	Alloh	Allah
2	Al Qur'an	Alquran
3	Al-Qur'an	Alquran
4	Al-Quran	Alquran
5	Azhar	Asar
6	Buda	Buddha
7	Budha	Buddha
8	February	Februari
9	Pebruari	Februari
10	Philipina	Filipina
11	Islamiyah	Islamiah
12	Itali	Italia
13	Jum'at	Jumat
14	Qodiriyah	Kadiriah
15	Qomariyah	Kamariah
16	Katholik	Katolik
17	Quraisy	Kuraisy
18	Mahapengasih	Maha Pengasih
19	Mahabarata	Mahabharata
20	Maha Esa	Mahaesa

Fig. 7. Potongan isi tabel pasangan_kata
Sumber: Dokumentasi pribadi

Algoritma KMP dan Booyer-Moore diimplementasikan dalam bentuk fungsi. Implementasi algoritma KMP terdiri dari dua fungsi, yaitu fungsi kmpMatch dan computeFail. Fungsi kmpMatch akan melakukan pencocokan *string* antara pola dan teks. Fungsi ini akan mengembalikan indeks karakter pertama pola pada teks jika ditemukan kecocokan dan mengembalikan -1 jika tidak ditemukan kecocokan. Kode program fungsi kmpMatch dapat dilihat pada Fig. 8.

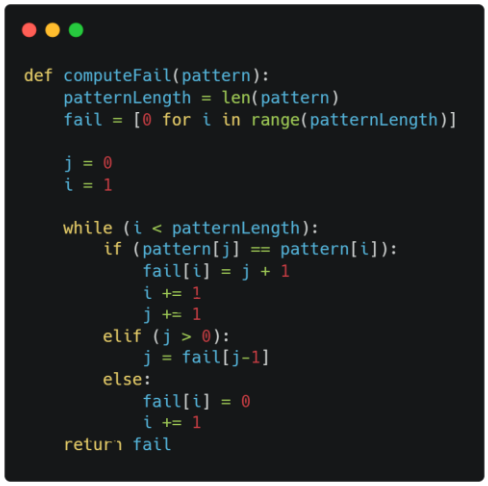
```
def kmpMatch(pattern, text):
    n = len(text)
    m = len(pattern)

    fail = computeFail(pattern)

    i = 0
    j = 0
    while (i < n):
        if (pattern[j] == text[i]):
            if (j == m - 1):
                return i - m + 1
            i += 1
            j += 1
        elif (j > 0):
            j = fail[j-1]
        else:
            i += 1
    return -1
```

Fig. 8. Kode fungsi kmpMatch
Sumber: Dokumentasi pribadi

Fungsi computeFail mengembalikan sebuah larik yang elemennya adalah hasil fungsi pinggiran untuk tiap karakter pada pola. Kode program fungsi computeFail dapat dilihat pada Fig. 9.



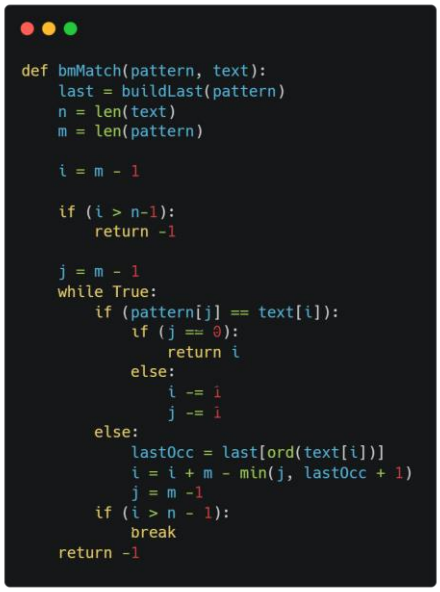
```
def computeFail(pattern):
    patternLength = len(pattern)
    fail = [0 for i in range(patternLength)]

    j = 0
    i = 1

    while (i < patternLength):
        if (pattern[j] == pattern[i]):
            fail[i] = j + 1
            i += 1
            j += 1
        elif (j > 0):
            j = fail[j-1]
        else:
            fail[i] = 0
            i += 1
    return fail
```

Fig. 9. Kode fungsi computeFail
Sumber: Dokumentasi pribadi

Implementasi algoritma Booyer-Moore juga terdiri dari dua fungsi, yaitu fungsi bmMatch dan buildLast. Fungsi bmMatch akan melakukan pencocokan *string* antara pola dan teks. Sama seperti fungsi kmpMatch, fungsi ini akan mengembalikan indeks karakter pertama pola pada teks jika ditemukan kecocokan dan mengembalikan -1 jika tidak ditemukan kecocokan. Kode program fungsi bmMatch dapat dilihat Fig. 10.



```
def bmMatch(pattern, text):
    last = buildLast(pattern)
    n = len(text)
    m = len(pattern)

    i = m - 1

    if (i > n-1):
        return -1

    j = m - 1
    while True:
        if (pattern[j] == text[i]):
            if (j == 0):
                return i
            else:
                i -= i
                j -= i
        else:
            lastOcc = last[ord(text[i])]
            i = i + m - min(j, lastOcc + 1)
            j = m -1
        if (i > n - 1):
            break
    return -1
```

Fig. 10. Kode fungsi bmMatch
Sumber: Dokumentasi pribadi

Fungsi buildLast mengembalikan larik yang berisi indeks kemunculan terakhir 127 karakter ASCII pada pola. Kode program fungsi computeFail dapat dilihat pada Fig. 11.

```
def buildLast(pattern):
    last = [-1 for i in range(128)]

    for i in range(len(pattern)):
        last[ord(pattern[i])] = i

    return last
```

Fig. 11. Kode fungsi bmMatch
Sumber: Dokumentasi pribadi

Pada program utama, pertama-tama program akan mengambil data kata baku dan tidak baku dari basis data menggunakan pustaka MariaDB. Data tersebut disimpan dalam sebuah *dictionary* atau yang biasa disebut dengan tipe data *map*. *Key* pada *dictionary* ini merupakan kata tidak baku dan *value*-nya merupakan kata bakunya. Selanjutnya, program akan menerima masukan nama file teks yang akan dicek penggunaan kata bakunya dan pilihan algoritma pencocokan *string*. Isi file teks lalu akan dibaca dan disimpan setiap barisnya dalam sebuah larik.

Pada tahap selanjutnya, program akan melakukan pencocokan *string* setiap baris teks terhadap setiap kemungkinan kata tidak baku yang ada pada *dictionary*. Potongan kode pada tahap ini dapat dilihat pada Fig. 12.

```
for i in range(len(lines)):
    for key in dictKata:
        if (algorithm == 1):
            result = kmpMatch(key.casefold(), lines[i].casefold())
        else:
            result = bmMatch(key.casefold(), lines[i].casefold())
        if (result != -1):
            oldWord = lines[i][result : result + len(key)]
            newWord = dictKata[key]
            if (oldWord[0].isupper()):
                newWord = newWord.capitalize()
            lines[i] = lines[i].replace(oldWord, newWord)
```

Fig. 12. Potongan kode pada tahap pencocokan *string* untuk setiap baris teks
Sumber: Dokumentasi pribadi

Pada Fig. 12 dapat dilihat bahwa fungsi kmpMatch dan bmMatch menerima argumen kata tidak baku dan teks. Pada dua argumen tersebut, dijalankan fungsi casefold. Fungsi ini berfungsi untuk mengubah seluruh karakter alfabet menjadi huruf kecil. Hal ini karena hamper seluruh daftar kata pada basis data tidak memedulikan kaidah penggunaan huruf kapital. Huruf pada suatu kata dapat kapital jika suatu kata berada di awal kalimat. Setelah fungsi kmpMatch atau bmMatch dijalankan, akan dilakukan pengecekan kembali terkait penggunaan huruf kapital. Jika kata yang dikoreksi mempunyai huruf kapital pada huruf pertamanya, huruf pertama kata baku yang akan menggantikannya akan dijadikan huruf kapital. Terakhir, kata yang tidak baku akan diganti dengan kata baku yang berkorespondensi.

IV. Hasil Pengujian

Program akan diuji menggunakan teks bahasa Indonesia yang mempunyai kesalahan penggunaan kata baku di dalamnya. Selain itu, eksekusi potongan kode pada Fig. 12 juga akan dihitung untuk melihat perbandingan waktu eksekusi antara algoritma KMP dan Booyer-Moore. Oleh karena itu, untuk setiap teks, program akan dijalankan beberapa kali untuk mendapatkan beberapa perbandingan waktu

A. Pengujian Teks Pertama

Pengujian teks pertama akan menggunakan file teks bahasa Indonesia di bawah:

> Pada hari Jum'at, saya pergi bersama adek saya ke apotik untuk membeli obat. Disain bangunan apotik itu sangat moderen. Setelah itu, saya dan adek saya pergi ke mesjid untuk melaksanakan sholat Jum'at. Mesjid ini sangat luas dan memiliki halaman seluas 1 hektar.

Pertama, program akan diuji menggunakan algoritma KMP. Keluaran program dengan algoritma KMP dapat dilihat pada Fig. 13.

```
Waktu eksekusi: 0.030964374542236328 s
Hasil pengoreksian kata tidak baku:
Pada hari Jumat, saya pergi bersama adik saya ke apotek untuk membeli obat.
Desain bangunan apotek itu sangat modern. Setelah itu, saya dan adik saya pergi ke masjid
untuk melaksanakan salat Jumat. Masjid ini sangat luas dan memiliki halaman seluas 1 hektare.
```

Fig. 13. Keluaran program dengan algoritma KMP pada pengujian teks pertama
Sumber: Dokumentasi pribadi

Selanjutnya, program akan diuji menggunakan algoritma Booyer-Moore. Keluaran program dengan algoritma Booyer-Moore dapat dilihat pada Fig. 14.

```
Waktu eksekusi: 0.027625083923339844 s
Hasil pengoreksian kata tidak baku:
Pada hari Jumat, saya pergi bersama adik saya ke apotek untuk membeli obat.
Desain bangunan apotek itu sangat modern. Setelah itu, saya dan adik saya pergi ke masjid
untuk melaksanakan salat Jumat. Masjid ini sangat luas dan memiliki halaman seluas 1 hektare.
```

Fig. 14. Keluaran program dengan algoritma Booyer-Moore pada pengujian teks pertama
Sumber: Dokumentasi pribadi

Pada kedua hasil pengujian, dapat dilihat bahwa terdapat sebelas kata tidak baku dan program dapat berhasil melakukan koreksi yang tepat. Namun, terlihat bahwa terdapat perbedaan waktu eksekusi program antara program yang menggunakan algoritma KMP dan Booyer-Moore. Program yang menggunakan algoritma KMP membutuhkan waktu eksekusi yang lebih lama dibandingkan program yang menggunakan algoritma Booyer-Moore. Agar lebih yakin, penulis melakukan lima kali pengujian menggunakan teks yang sama dan didapatkan hasil seperti yang ditunjukkan pada Tabel I.

TABEL I. Hasil Pengujian Teks Pertama

Nomor	Waktu Eksekusi (detik)	
	KMP	***Booyer-Moore***
1	0.030964374542236328	0.027625083923339844
2	0.03092336654663086	0.028305530548095703
3	0.03195524215698242	0.03136944770812988
4	0.0331952571868896	0.026986122131347656
5	0.03362751007080078	0.028885602951049805

Pada Tabel I, terlihat bahwa algoritma KMP waktu eksekusi yang lebih lama daripada algoritma Booyer-Moore untuk setiap nomor. Oleh karena itu, dapat disimpulkan bahwa algoritma Booyer-Moore lebih cepat daripada algoritma KMP pada pengujian teks pertama.

B. Pengujian Teks Kedua

Pengujian teks kedua akan menggunakan file teks bahasa Indonesia di bawah:

> Budi adalah seorang siswa jenius yang akan mendaftar SBMPTN. Oleh karena itu, dia pergi untuk melakukan photo copy beberapa berkas. Selanjutnya, dia akan membuat pas foto. Maka dari itu, dia berpakaian formil hari ini.

Sama seperti pengujian teks pertama, program akan diuji terlebih dahulu menggunakan algoritma KMP. Keluaran program dengan algoritma KMP dapat dilihat pada Fig. 15.

```
Waktu eksekusi: 0.02790236473083496 s
Hasil pengoreksian kata tidak baku:
Budi adalah seorang siswa genius yang akan mendaftar SBMPTN. Oleh karena itu,
dia pergi untuk melakukan fotokopi beberapa berkas. Selanjutnya, dia akan
membuat pasfoto. Maka dari itu, dia berpakaian formal hari ini.
```

Fig. 15. Keluaran program dengan algoritma KMP pada pengujian teks kedua
Sumber: Dokumentasi pribadi

Selanjutnya, program akan diuji menggunakan algoritma Booyer-Moore. Keluaran program dengan algoritma Booyer-Moore dapat dilihat pada Fig. 16.

```
Waktu eksekusi: 0.02399468421936035 s
Hasil pengoreksian kata tidak baku:
Budi adalah seorang siswa genius yang akan mendaftar SBMPTN. Oleh karena itu,
dia pergi untuk melakukan fotokopi beberapa berkas. Selanjutnya, dia akan
membuat pasfoto. Maka dari itu, dia berpakaian formal hari ini.
```

Fig. 16. Keluaran program dengan algoritma Booyer-Moore pada pengujian teks kedua
Sumber: Dokumentasi pribadi

Pada kedua hasil pengujian, dapat dilihat bahwa terdapat empat kata tidak baku kedua algoritma berhasil memberikan hasil yang tepat. Algoritma KMP membutuhkan waktu eksekusi yang lebih lama seperti pada pengujian teks pertama. Pada pengujian teks ini juga akan dilakukan lima kali pengujian. Hasil lima kali pengujian teks ini ditunjukkan pada Tabel II.

TABEL II. Hasil Pengujian Teks Kedua

Nomor	Waktu Eksekusi (detik)	
	KMP	***Booyer-Moore***
1	0.02790236473083496	0.02399468421936035
2	0.026868343353271484	0.026311874389648438
3	0.025613069534301758	0.024353504180908203
4	0.02770256996154785	0.02460765838623047

5	0.028172016143798828	0.025269031524658203

Terlihat pada tabel II, algoritma KMP juga membutuhkan waktu eksekusi yang lebih lama untuk setiap nomor seperti pengujian teks pertama. Maka dari itu, didapatkan kesimpulan untuk pengujian teks kedua bahwa algoritma Booyer-Moore lebih cepat daripada algoritma KMP.

C. Pengujian Teks Ketiga

Pengujian teks ketiga akan menggunakan file teks bahasa Indonesia di bawah:

> Pak Budi memiliki banyak aktifitas. Dengan hanya menggunakan sendal dan kaos, beliau melakukan faksinasi. Setelah itu, beliau pulang naik bis. Lalu, beliau langsung menemui clien untuk membahas projek bulan Pebruari.

Sama seperti kedua pengujian sebelumnya, program akan diuji menggunakan algoritma KMP terlebih dahulu. Keluaran program dengan algoritma KMP dapat dilihat pada Fig. 17.

```
Waktu eksekusi: 0.0280759334564209 s
Hasil pengoreksian kata tidak baku:
Pak Budi memiliki banyak aktivitas. Dengan hanya menggunakan sandal dan kaus,
beliau melakukan vaksinasi. Setelah itu, beliau pulang naik bus. Lalu, beliau
langsung menemui klien untuk membahas proyek bulan Februari.
```

Fig. 17. Keluaran program dengan algoritma KMP pada pengujian teks ketiga
Sumber: Dokumentasi pribadi

Program juga akan diuji menggunakan algoritma Booyer-Moore. Keluaran program dengan algoritma Booyer-Moore dapat dilihat pada Fig. 18.

```
Waktu eksekusi: 0.024987220764160156 s
Hasil pengoreksian kata tidak baku:
Pak Budi memiliki banyak aktivitas. Dengan hanya menggunakan sandal dan kaus,
beliau melakukan vaksinasi. Setelah itu, beliau pulang naik bus. Lalu, beliau
langsung menemui klien untuk membahas proyek bulan Februari.
```

Fig. 18. Keluaran program dengan algoritma Booyer-Moore pada pengujian teks ketiga
Sumber: Dokumentasi pribadi

Pengujian teks ketiga juga berhasil melakukan koreksi yang tepat pada delapan kata tidak baku. Pada pengujian ini, algoritma KMP juga membutuhkan waktu eksekusi yang lebih lama dibandingkan program yang menggunakan algoritma Booyer-Moore sama seperti dua pengujian sebelumnya. Hasil lima kali pengujian dapat dilihat pada Tabel III.

TABEL III. HASIL PENGUJIAN TEKS KETIGA

Nomor	Waktu Eksekusi (detik)	
	KMP	***Booyer-Moore***
1	0.02888941764831543	0.024987220764160156
2	0.02947854995727539	0.0244143009185791
3	0.026393413543701172	0.025290727615356445
4	0.02694559097290039	0.025189876556396484
5	0.02691173553466797	0.025684118270874023

Tabel III memperlihatkan bahwa algoritma KMP memiliki waktu eksekusi yang lebih lama daripada algoritma Booyer-Moore untuk setiap nomor. Maka dari itu, kesimpulan yang didapatkan sama dengan pengujian pertama dan kedua, yaitu algoritma Booyer-Moore lebih cepat daripada algoritma KMP.

V. KESIMPULAN DAN SARAN

Berdasarkan hasil pengujian, algoritma KMP dan Booyer-Moore memiliki perbedaan waktu eksekusi. Pada setiap teks pengujian, algoritma Booyer-Moore memiliki waktu eksekusi yang lebih cepat daripada algoritma KMP. Hal ini karena algoritma Booyer-Moore cocok digunakan untuk jumlah alfabet yang besar, seperi bahasa Indonesia atau bahasa Inggris. Sebaliknya, algoritma Booyer-Moore berjalan lambat jika digunakan pada jumlah alfabet yang sedikit, misalnya bilangan biner. Di sisi lain, algoritma KMP lebih cocok digunakan untuk jumlah alfabet yang sedikit, seperti bilangan biner. Algoritma KMP tidak cocok digunakan untuk jumlah alfabet yang besar karena kemungkinan untuk terjadi ketidakcocokan pada awal pencocokan lebih besar.

Terlepas dari hal tersebut, kedua algoritma berhasil mengoreksi kata tidak baku dengan tepat dengan tiga teks pengujian. Kedua algoritma ini melakukan pencocokan antara kata tidak baku pada basis data dengan teks yang diberikan. Namun, program yang dibuat masih belum sempurna. Program belum menangani kasus seperti kata tidak baku yang menjadi *subset* pada sebuah kata berimbuhan. Contohnya adalah kata "mempunyai" yang merupakan *subset* kata "mpu". Kata "mpu" merupakan kata tidak baku dari "empu". Karena pada basis data terdapat kata "mpu" sebagai kata tidak baku, kata "mempunyai" akan dikoreksi program menjadi kata "meempunyai". Tentu saja hal ini membuat penulisan kata yang sudah benar menjadi salah. Maka dari itu, program perlu disempurnakan lebih lanjut.

PRANALA VIDEO YOUTUBE

https://youtu.be/9wKit5hKjvQ

UCAPAN TERIMA KASIH

Pertama-tama, penulis mengucapkan terima kasih kepada Tuhan Yang Maha Esa karena atas rahmat dan karunia-Nya, penulis bisa menyelesaikan penulisan makalah ini. Penulis juga berterima kasih kepada orang tua dan keluarga penulis karena telah memberi dukungan terhadap pelaksanaan kuliah penulis. Selain itu, penulis juga mengucapkan terima kasih kepada Ibu Dr. Masayu Leylia Khodra, S.T., M.T., Ibu Dr. Nur Ulfa Maulidevi, S.T., M.Sc., dan Bapak Dr. Ir. Rinaldi Munir, M.T., atas bimbingan dan ilmu yang telah diberikan selama pelaksaan kuliah Strategi Algoritma semester II tahun ajaran 2021/2022.

REFERENSI

[1] P. E. Black, "String Matching". Dictionary of Algorithms and Data Structures. https://xlinux.nist.gov/dads/HTML/stringMatching.html (diakses pada 20 Mei 2022).

[2] R. Munir, “Pencocokan String”. Homepage Rinaldi Munir. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf (diakses pada 20 Mei 2022).

[3] A. Haryanto, “Kata Baku dan Tidak Baku: Pengertian, Fungsi, Ciri & Contohnya”. Tirto.id. https://tirto.id/kata-baku-dan-tidak-baku-pengertian-fungsi-ciri-contohnya-gjie (diakses pada 20 Mei 2022).

[4] S. Nurdjan, Firman, Mirnawati, *Bahasa Indonesia untuk Perguruan Tinggi*. Makassar: Aksara Timur, 2016. pp. 35-36.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 21 Mei 2022

Mohamad Daffa Argakoesoemah 13520118