Pengaplikasian Algoritma Boyer-Moore dan Regular Expression dalam Membangun Mesin Pencarian Menu dan Harga Makanan pada Aplikasi Desktop

Eiffel Aqila Amarendra - 13520074
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13520074@std.stei.itb.ac.id

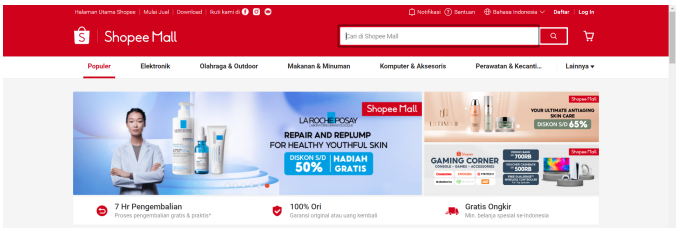
***Abstrak*—Dewasa ini, teknologi merupakan hal yang tidak dapat terlepas dari kehidupan manusia, seperti mesin pencarian yang sudah menjadi hal krusial di dalam kehidupan. Mesin pencarian dapat ditemukan di berbagai hal, seperti Google dan aplikasi belanja daring/layanan pesan antar. Pada makalah ini dibahas seputar mesin pencarian menu dan harga kuliner berdasarkan masukan query dengan menggunakan algoritma *Boyer-Moore* sebagai algoritma untuk melakukan pencarian data dan *Regular Expression* sebagai pemvalidasi masukan pada aplikasi desktop.**

Kata kunci—mesin pencarian; pencocokan string; boyer-moore; regular expression; aplikasi desktop

I. Pendahuluan

Di era digitalisasi ini, teknologi merupakan suatu komponen yang tidak dapat terlepas dari kehidupan manusia. Hal tersebut didukung dengan fakta bahwa teknologi telah memberikan kemudahan, kenyamanan, serta kesempatan terhadap penggunanya di dalam berbagai aspek di kehidupan, mulai dari aspek sosial, ekonomi, pendidikan, hingga industri sesuai dengan kebutuhan pengguna teknologi tersebut. Selain itu, teknologi juga telah memberikan kemudahan bagi penggunanya untuk dapat memperoleh dan memenuhi kebutuhan informasinya. Oleh karena itu, mesin pencarian sudah menjadi hal yang krusial di dalam kehidupan.

Mesin pencarian (*search engine*) merupakan salah satu perangkat yang dapat digunakan pengguna untuk menemukan informasi yang sesuai berdasarkan parameter masukan. Dengan banyaknya informasi yang bertebaran di internet dan sebuah basis data, mesin pencarian menyajikan informasi yang dibutuhkan oleh pengguna dengan lebih mudah, singkat, dan sesuai kebutuhan. Mesin pencarian dapat ditemukan di berbagai hal, seperti pada Google dan aplikasi belanja daring/layanan pesan antar seperti Shopee dan GoFood.



Gambar 1. Mesin Pencarian pada Shopee

(Sumber: https://shopee.co.id/mall/ , diakses pada 22 Mei 2022)

Makalah ini dibuat oleh penulis dengan tujuan untuk membantu para pembaca dalam melakukan pencarian dengan menggunakan mesin pencarian menu dan harga kuliner berdasarkan masukan *query* dengan menggunakan algoritma *Boyer-Moore* dan *Regular Expression* pada aplikasi desktop.

II. Landasan Teori

A. Pencocokan String

String merupakan sebuah tipe data yang berisi kumpulan beberapa karakter, seperti huruf, angka, spasi, dan karakter-karakter lainnya. Sebuah String S dengan panjang m berikut

$$S = x_0 x_1 \ldots x_m$$

memiliki *prefix* atau awalan, yakni $S[0 \ldots k]$ dan *suffix* atau akhiran, yakni $S[k \ldots m - 1]$ dengan k terletak di antara 0 hingga $m - 1$.

Pencocokan String/*String Matching*/*Pattern Matching* merupakan pencarian sebuah *pattern*/String P dengan panjang m di dalam sebuah teks T dengan panjang n dengan n lebih besar dari pada m. Pencocokan String merupakan persoalan aplikatif yang dapat dengan mudah ditemukan di berbagai aspek di dalam kehidupan kita. Hal ini dapat ditemukan dalam fitur pencarian di dalam *Text Editor*, *Web Search Engine*, Analisis Citra, serta pencocokan rantai asam amino pada rantai DNA (*Bioinformatics*).

Terdapat beberapa algoritma yang dapat menyelesaikan persoalan pencocokan String tersebut, antara lain algoritma *Brute Force*, *Knuth-Morris-Pratt* (KMP), *Boyer-Moore*, serta *Regular Expression* (Regex). Masing-masing algoritma memiliki kelebihan dan kekurangannya masing-masing tergantung persoalan yang diberikan.

B. Algoritma Brute Force

Algoritma *Brute Force* merupakan algoritma penyelesaian suatu persoalan dengan cara yang sederhana, langsung, dan jelas. Pada persoalan pencocokan String, algoritma *brute force* memeriksa setiap posisi dari teks *T* yang mulai dari indeks ke-0 hingga indeks ke-(*m* - *n*) hingga ditemukan karakter pada teks *T* yang sesuai dengan karakter pertama *pattern P*. Jika ditemukan karakter yang sama, pemeriksaan dilanjutkan dengan indeks selanjutnya dari *pattern P* dan teks *T* hingga ditemukan perbedaan atau didapatkan solusi. Jika ditemukan perbedaan, *pattern* bergerak 1 karakter ke kanan hingga ditemukan kembali persamaan antara karakter pada teks *T* dan karakter pertama *pattern P* tanpa peduli adanya pengulangan pola atau tidak. Hal tersebut berulang hingga ditemukan solusi atau sudah mencapai ujung.

Pencocokan String dengan menggunakan algoritma *Brute Force* menghasilkan kompleksitas waktu $O(mn)$ untuk kasus terburuk, $O(n)$ untuk kasus terbaik, dan $O(m + n)$ untuk kasus rerata. Algoritma ini efisien ketika variansi karakter tinggi, misalnya karakter A..Z dan tidak efisien ketika variansi karakter rendah, misalnya bilangan biner 0 dan 1.

C. Algoritma Knuth-Morris-Pratt

Algoritma *Knuth-Morris-Pratt* (KMP) merupakan algoritma pencocokan string yang cenderung lebih cerdas dibandingkan *brute force*. Algoritma ini melakukan pergeseran dari kiri ke kanan, namun dengan beberapa pertimbangan tertentu. Ketika ditemukan perbedaan/ketidakcocokan karakter pada saat dilakukan pencocokan, algoritma KMP akan melakukan penggeseran ke kanan terbesar sehingga perbandingan-perbandingan yang tidak perlu dapat dihindari, sedangkan pada algoritma *brute force* hal tersebut tidak dilakukan.

Algoritma KMP melakukan pra-pemrosesan *pattern* P dengan fungsi pinggiran (*KMP Border Function*) $b(k)$ untuk memperoleh prefiks $P[0 .. k]$ terbesar yang merupakan sufiks $P[1 .. k]$. Ketika ditemukan ketidakcocokan pada *pattern* P indeks ke-j, j akan bergerak ke kanan sehingga $j = b(j - 1)$.

Pencocokan String dengan menggunakan algoritma KMP menghasilkan waktu $O(m)$ untuk menentukan fungsi pinggiran dan $O(n)$ untuk melakukan pencarian string sehingga kompleksitas waktu untuk algoritma KMP adalah sebesar $O(m+n)$. Hal tersebut membuktikan bahwa algoritma KMP cenderung lebih atau bahkan sangat cepat dibandingkan dengan algoritma *brute force*.

Algoritma KMP efisien ketika memproses file yang sangat besar karena algoritma karena tidak perlu adanya pergerakan mundur pada masukan teks *T*. Namun, algoritma ini kurang cocok ketika variansi karakter tinggi karena peluang terjadinya ketidakcocokan awal menjadi sangat tinggi.

D. Algoritma Boyer-Moore

Algoritma *Boyer-Moore* merupakan algoritma pencocokan string yang menggunakan dua buah teknik, antara lain *the looking-glass technique* dan *the character-jump technique*. *The looking-glass technique* merupakan teknik pencarian *pattern* pada teks dengan bergerak mundur dengan memulai dari bagian akhir. Sedangkan, *the character-jump technique* merupakan teknik pergeseran yang dilakukan ketika terjadi ketidakcocokan karakter pada saat pencocokan.

Pergeseran yang dilakukan oleh *the character-jump technique* dibagi menjadi tiga buah kasus. Kasus pertama terjadi ketika terdapat perbedaan karakter *pattern P* dengan *teks T* pada suatu indeks, namun karakter pada *teks T* di indeks tersebut terdapat pula pada *pattern P* di kiri indeks *P* sekarang, pergeseran *pattern P* ke kanan dilakukan sehingga karakter tersebut pada *pattern P* dan teks *T* menjadi sejajar. Kasus kedua terjadi ketika terdapat perbedaan karakter *pattern P* dengan *teks T* pada suatu indeks, namun karakter pada *teks T* di indeks tersebut terdapat pula pada *pattern P* di kanan indeks *P* sekarang, pergeseran *pattern P* ke kanan dilakukan sebanyak 1 karakter. Kasus ketiga terjadi ketika kedua kasus di atas tidak dipenuhi, pergeseran *pattern P* ke kanan dilakukan sehingga $P[0]$ sejajar dengan $T[i + 1]$.

Algoritma *Boyer-Moore* melakukan pra-pemrosesan *pattern P* dan alfabet *A* dengan fungsi *last occurrence* untuk memetakan setiap alfabet ke dalam indeks-indeks integer pada array *L*(). Fungsi *last occurence* $L(x)$ menyimpan indeks terbesar ketika $P[i] == x$ atau -1 ketika tidak ditemukan.

Pencocokan String dengan menggunakan algoritma *Boyer-Moore* menghasilkan waktu $O(mn + A)$ untuk kasus terburuk. Namun, algoritma ini efisien ketika memproses string dengan variansi karakter yang tinggi dan tidak efisien ketika variansi karakter rendah layaknya karakteristik algoritma *brute force*. Meskipun demikian, algoritma *boyer-moore* cenderung lebih atau bahkan sangat cepat dibandingkan dengan algoritma *brute force*.

E. Regular Expression

Regular Expression (regex) merupakan notasi standar yang digunakan pada persoalan pencocokan string. Regex merupakan sebuah sekuens karakter yang mendeskripsikan suatu *search pattern*. Regex dapat dimanfaatkan untuk memeriksa apakan di dalam sebuah teks terdapat atau terkandung sebuah *search pattern* yang telah dispesifikasi oleh regex terkait.

Dilansir dari w3schools.com, di dalam regex pada python dikenal beberapa fungsi dan istilah-istilah seperti *metacharacters*, *special sequences*, dan *sets*. *Metacharacters* merupakan kumpulan karakter yang memiliki arti khusus di dalam *regular expression*. *Special sequences* merupakan karakter yang diawali "\" yang memiliki arti khusus dalam

perincian karakter. *Sets* merupakan kumpulan karakter yang berada di antara "[]".

Tabel 1. Metacharacters

Sumber: https://www.w3schools.com/

Karakter	Deskripsi
[]	Set karakter
\	Digunakan sebagai penanda *special sequence*
.	Seluruh karakter kecuali newline
^	Dimulai dengan
$	Diakhiri dengan
*	Nol atau lebih kemunculan
+	Satu atau lebih kemunculan
?	Nol atau satu kemunculan
{}	Perincian jumlah kemunculan
|	Atau
()	*Capture* atau *group*

Tabel 2. Special Sequences

Sumber: https://www.w3schools.com/

Karakter	Deskripsi
\A	Memeriksa apakah karakter terletak di awal *string*
\b	Memeriksa apakah karakter terletak di awal atau akhir sebuah kata
\B	Memeriksa apakah apakah karakter ditemukan dan tidak terletak di awal atau akhir sebuah kata
\d	Memeriksa apakah karakter adalah digit (0–9)
\D	Memeriksa apakah karakter bukan digit
\s	Memeriksa apakah karakter adalah *whitespace*
\S	Memeriksa apakah karakter bukan *whitespace*
\w	Memeriksa apakah string mengandung *word characters* (a–Z, 0–9, _)
\W	Memeriksa apakah string tidak mengandung *word characters*
\Z	Memeriksa apakah karakter terletak di akhir *string*

Tabel 3. Sets

Sumber: https://www.w3schools.com/

Set	Deskripsi
[arn]	Memeriksa apakah terdapat salah satu karakter a, r, atau n
[a-n]	Memeriksa apakah terdapat salah satu karakter di antara a–n (huruf kecil)
[^arn]	Memeriksa apakah tidak terdapat salah satu karakter a, r, atau n
[123]	Memeriksa apakah terdapat salah satu digit 1, 2, atau 3
[1–3]	Memeriksa apakah terdapat salah satu digit di antara 1–3
[0–5][0–9]	Memeriksa apakah terdapat salah satu digit di antara 00–59
[a-zA-Z]	Memeriksa apakah terdapat salah satu karakter di antara a–z (huruf kecil) atau A–Z (huruf kapital)
[+]	Memeriksa apakah terdapat karakter +

F. Kuliner

Kuliner merupakan produk hasil olahan yang dapat berupa makanan maupun minuman. Secara etimologis, kuliner berasal dari kata dalam bahasa inggris *'culinary'* yang diterjemahkan menjadi bahasa indonesia. Kata '*culinary*' berasal dari kata dalam bahasa latin '*culina*' yang berarti sebuah ruang memasak makanan. Oleh karena itu, kuliner juga dapat didefinisikan sebagai sesuatu yang berhubungan dengan proses memasak.

III. Pembahasan

Untuk membangun mesin pencarian menu dan harga makanan, penulis membuat terlebih dahulu sebuah basis data `kulinerfinder` sebagai data uji yang berisi data sebagai berikut,

Tabel 4. Isi Data Uji

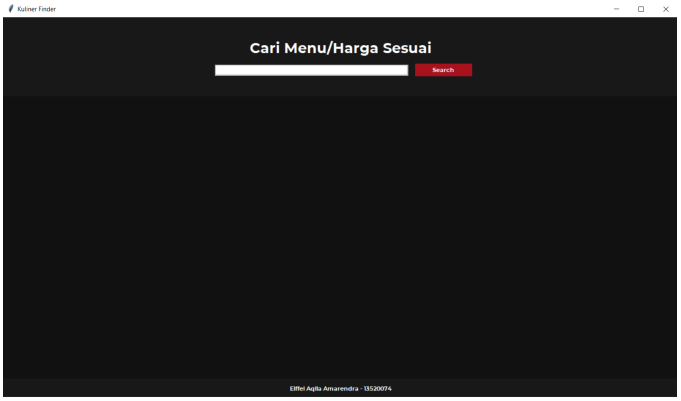
Sumber: Dokumentasi Pribadi

No	Nama	Harga

1	Ayam Goreng	10000
2	Ayam Bakar	10000
3	Ayam Balado	12000
4	Ayam Gulai	12000
5	Ikan Gurame Goreng Asam Manis	54000
6	Ikan Gurame Goreng Padang	56000
7	Ikan Gurame Bakar	52000
8	Ikan Kerapu Goreng Asam Manis	52000
9	Ikan Kerapu Goreng Padang	54000
10	Ikan Kerapu Bakar	20000

Berdasarkan data pada tabel tersebut, dapat terlihat bahwa data tersebut terdiri dari atribut nama dan harga. Atribut-atribut tersebut akan menjadi parameter pencocokan String berdasarkan masukan dari pengguna.

Penulis kemudian membangun sebuah *desktop application* sederhana yang menggunakan bahasa pemrograman Python serta Tkinter sebagai Framework GUI. Pada aplikasi tersebut, pengguna dapat memasukan query, baik berupa nama makanan, harga makanan, atau keduanya. Kemudian, aplikasi akan menampilkan daftar menu yang sesuai dengan kriteria yang diinginkan/dibutuhkan oleh pengguna.



Gambar 2. Tampilan Awal Aplikasi

Sumber: Dokumentasi Pribadi

Pada aplikasi ini, penulis menggunakan dua jenis algoritma pencocokan String, antara lain *Regular Expression* sebagai pemvalidasi masukan pengguna dan pemeriksa apakah masukan pengguna berupa nama makanan, harga makanan, atau keduanya serta algoritma *Boyer-Moore* sebagai algoritma untuk melakukan pencarian data dengan membandingkan antara teks dari setiap data pada basis data dan *pattern*(*-pattern*) masukan pengguna.

Penulis memilih untuk menggunakan algoritma *Boyer-Moore* untuk melakukan pencarian data dibandingkan algoritma *brute force* dan *knuth-morris-pratt* karena algoritma ini sangat cocok untuk memproses string dengan variansi karakter yang tinggi. Misalnya, pada string "Ayam Goreng" terbukti bahwa alfabet yang digunakan pada string sangat bervariasi.

Untuk memeriksa dan memvalidasi masukan pengguna, penulis membuat satu buah fungsi yang memanfaatkan *regular expression*, yakni sebagai berikut

```
def validateQuery(self, query, datas):
    dataFiltered = []

    # Query Type
    nameOnly = re.compile(r"\w\w*")
    priceOnly = re.compile(r"\d\d*")
    namePrice = re.compile(r"\w\w* \d\d*")

    if (namePrice.fullmatch(query)):
        queries = query.split()
        for data in datas:
            if (self.matchBM(data[0], queries[0]) and
(int(data[1]) == int(queries[1]))):
                dataFiltered.append(data)

        if (len(dataFiltered) > 0):
            self.displayItemFrames(self.frame,
dataFiltered, 0)
        else:
            messagebox.showinfo("Error", "No data
found")

    elif (priceOnly.fullmatch(query)):
        for data in datas:
            if ((int(data[1]) == int(query))):
                dataFiltered.append(data)

        if (len(dataFiltered) > 0):
            self.displayItemFrames(self.frame,
dataFiltered, 0)
        else:
            messagebox.showinfo("Error", "No data
found")

    elif (nameOnly.fullmatch(query)):
        for data in datas:
            if (self.matchBM(data[0], query)):
                dataFiltered.append(data)

        if (len(dataFiltered) > 0):
            self.displayItemFrames(self.frame,
dataFiltered, 0)
        else:
            messagebox.showinfo("Error", "No data
found")
    else:
        messagebox.showinfo("Error", "Invalid input: "
+ query)
```

Pada fungsi *validateQuery* tersebut, penulis membagi masukan valid menjadi tiga buah kasus, antara lain masukan nama menu berupa *alphanumeric* sepanjang 1 kata, masukan harga makanan berupa bilangan integer, serta masukan nama menu berupa *alphanumeric* sepanjang 1 kata dan harga makanan berupa bilangan integer. Sebagai contoh, masukan

yang valid, yakni 'Ikan', '50000', 'Ikan 50000' dan masukan yang tidak valid, yakni '', '50000 Ikan', 'Ikan Goreng'.

Jika masukan pengguna tidak valid, program akan mengirimkan pesan 'Invalid Input' atau 'Query is empty' jika masukan pengguna kosong ke layar. Jika masukan pengguna valid, program akan memproses masukan pengguna tersebut untuk dilakukan pencarian data berdasarkan *query* masukan pengguna dengan memanggil fungsi *matchBM* untuk pencarian menu (*alphanumeric*) dan perbandingan integer biasa untuk pencarian harga (integer).

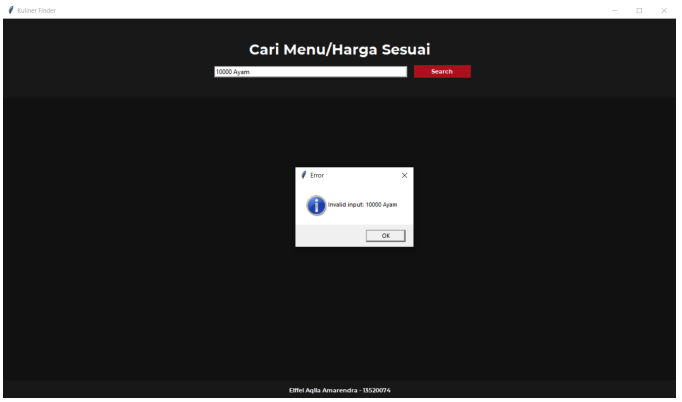
Untuk melakukan pencarian data berdasarkan *query* pengguna, penulis membuat dua buah fungsi yang memanfaatkan algoritma *boyer-moore*, yakni sebagai berikut

```
def matchBM(self, text, pattern):
    lastOccurrence = self.buildLast(pattern)
    n = len(text)
    m = len(pattern)
    # Pattern lebih panjang dari text
    if (m > n):
        return False
    else:
        i = m - 1
        j = m - 1
        while (i <= n - 1):
            if (pattern[j] == text[i]):
                if (j == 0):
                    return True
                else:
                    j -= 1
                    i -= 1
            else:
                lo = lastOccurrence[ord(text[i])]
                i = i + m - min(j, lo+1)
                j = m - 1
    return False

def buildLast(self, pattern):
    # Penyimpanan index last occurrence
    lastOccurrence = []
    for i in range (128):
        lastOccurrence.append(-1)

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

    return lastOccurrence
```

Setelah pencarian data telah dilakukan, fungsi *matchBM* akan mengembalikan boolean True jika ditemukan kesamaan pada *teks* terkait atau False jika tidak. Jika ditemukan adanya data yang cocok dengan *query* masukan pengguna, program akan menampilkan data menu, harga, dan gambar kuliner yang sesuai dengan kriteria pengguna ke layar. Jika tidak ditemukan, program akan mengirimkan pesan 'No data found' ke layar.

Untuk menjalankan program, jalankan program python sehingga muncul tampilan awal program seperti pada gambar 2. Pada form pencarian, masukan *query* nama, harga, atau keduanya yang valid untuk melakukan pencarian kuliner yang sesuai dengan kriteria. Kemudian, tekan tombol "Search". Berikut merupakan beberapa hasil pengujian program "KulinerFinder".

Pengujian pertama adalah pengujian untuk masukan *query* yang tidak valid. Untuk pengujian ini, *query* yang dimasukan adalah '10000 Ayam'. Hasil tangkapan layar pengujian tersebut adalah sebagai berikut.

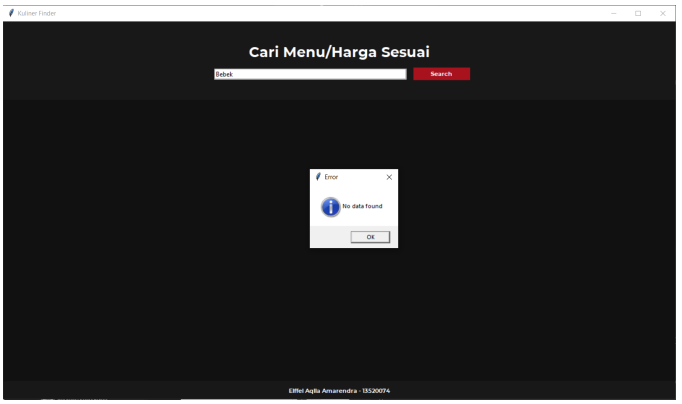


Gambar 3. Kasus: Query yang Tidak Valid

Sumber: Dokumentasi Pribadi

Berdasarkan pengujian di atas, program mengklasifikasikan *query* tersebut sebagai *query* yang tidak valid. Masukan *query* yang valid, sebagaimana telah dijelaskan sebelumnya, antara lain masukan nama menu berupa *alphanumeric* sepanjang 1 kata, masukan harga makanan berupa bilangan integer, serta masukan nama menu berupa *alphanumeric* sepanjang 1 kata dan harga makanan berupa bilangan integer.

Pengujian kedua adalah pengujian untuk masukan *query* valid untuk mencari data yang tidak tersedia di dalam basis data. Untuk pengujian ini, *query* yang dimasukan adalah 'Bebek'. Hasil tangkapan layar pengujian tersebut adalah sebagai berikut.

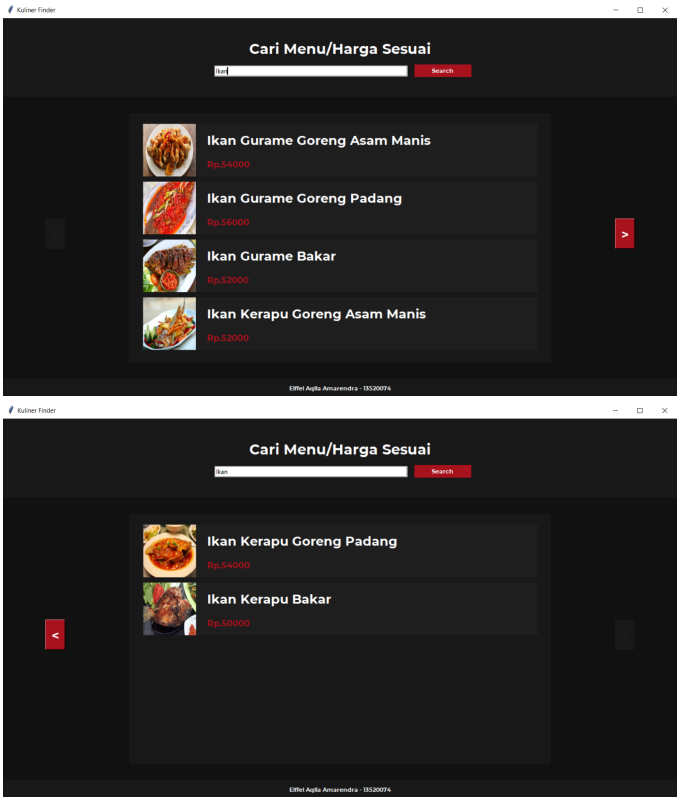


Gambar 4. Kasus: Data Tidak Ditemukan

Sumber: Dokumentasi Pribadi

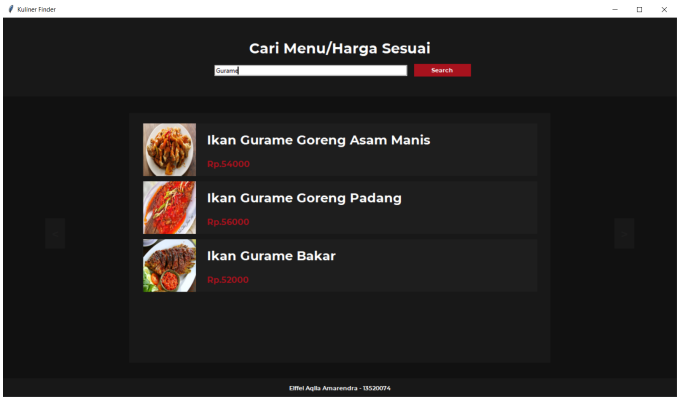
Berdasarkan pengujian di atas, *query* merupakan dan diklasifikasikan sebagai *query* yang valid, namun karena pada basis data tidak ada menu yang berisi kata 'Bebek' di dalamnya, program menampilkan pesan 'No data found'.

Pengujian ketiga adalah pengujian untuk masukan *query* valid berupa menu makanan dengan tipe *alphanumeric* untuk mencari data yang tersedia di dalam basis data. Untuk pengujian ini, *query* yang dimasukan adalah 'Ikan' dan 'Gurame'. Hasil tangkapan layar pengujian tersebut adalah sebagai berikut.



Gambar 5. Kasus: Pencarian Hanya Nama: Ikan

Sumber: Dokumentasi Pribadi

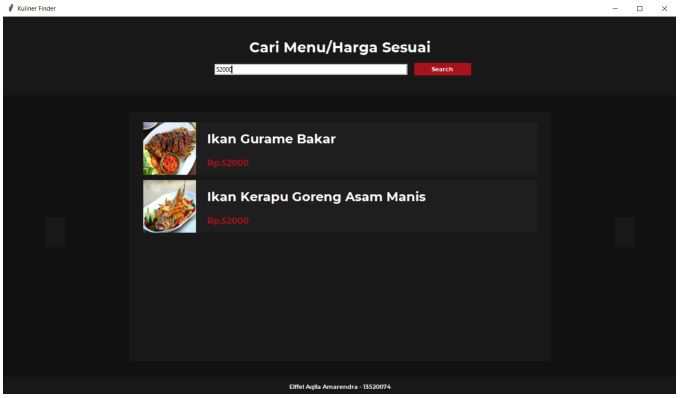


Gambar 6. Kasus: Pencarian Hanya Nama: Gurame

Sumber: Dokumentasi Pribadi

Berdasarkan pengujian di atas, *query* merupakan dan diklasifikasikan sebagai *query* yang valid serta ditemukan menu dengan kata kunci serupa pada basis data. Pada pengujian tersebut, terbukti bahwa pencocokan *query* akan dilakukan pada setiap kata di dalam *string* data sehingga dimanapun letak *query* pada *string* baik di awal, tengah, atau akhir kata pada *string*, hal tersebut akan diterima. Selain itu, terbukti pula bahwa program dapat menemukan data berdasarkan masukan *alphanumeric* 1 kata yang telah tervalidasi *regular expression*.

Pengujian keempat adalah pengujian untuk masukan *query* valid berupa harga makanan dengan tipe bilangan integer untuk mencari data yang tersedia di dalam basis data. Untuk pengujian ini, *query* yang dimasukan adalah '52000'. Hasil tangkapan layar pengujian tersebut adalah sebagai berikut.

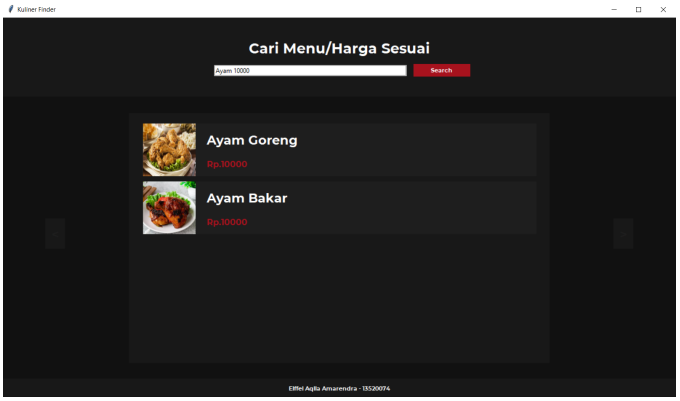


Gambar 7. Kasus: Pencarian Hanya Harga: 52000

Sumber: Dokumentasi Pribadi

Berdasarkan pengujian di atas, *query* merupakan dan diklasifikasikan sebagai *query* yang valid serta ditemukan menu dengan kata kunci serupa pada basis data. Pada pengujian tersebut, terbukti bahwa program dapat menemukan data berdasarkan masukan *integer* yang telah tervalidasi *regular expression*.

Pengujian kelima adalah pengujian untuk masukan *query* valid berupa menu makanan dengan tipe *alphanumeric* dan harga makanan dengan tipe bilangan integer untuk mencari data yang tersedia di dalam basis data. Untuk pengujian ini, *query* yang dimasukan adalah 'Ayam 10000'. Hasil tangkapan layar pengujian tersebut adalah sebagai berikut.



Gambar 8. Kasus: Pencarian Nama dan Harga: Ayam 10000

Sumber: Dokumentasi Pribadi

Berdasarkan pengujian di atas, *query* merupakan dan diklasifikasikan sebagai *query* yang valid serta ditemukan menu dan harga dengan kata kunci serupa pada basis data. Pada pengujian tersebut, terbukti pula bahwa program dapat menemukan data berdasarkan masukan *alphanumeric* 1 kata dan masukan *integer* yang telah tervalidasi *regular expression*.

Berdasarkan hasil pengujian-pengujian di atas, terbukti bahwa algoritma *boyer-moore* dan *regular expression* pada program penulis dapat berjalan dengan baik dan sesuai dengan yang diharapkan.

IV. Kesimpulan

Pencocokan string merupakan pencarian sebuah *string/pattern* di dalam teks. Pencocokan string memiliki berbagai macam algoritma, seperti algoritma *brute force*, *knuth-morris-pratt*, *boyer-moore*, dan *regular expression*. Pencocokan string berpotensi untuk diaplikasikan untuk menjadi solusi dalam berbagai persoalan, salah satunya dengan memanfaatkan algoritma *boyer-moore* sebagai algoritma untuk melakukan pencarian data dan *regular expression* sebagai pemvalidasi masukan dalam mesin pencarian menu dan harga makanan pada aplikasi *desktop*.

Program mesin pencarian menu dan harga makanan yang telah dibuat penulis dapat berjalan dengan baik dan sesuai dengan yang diharapkan. Meskipun demikian, masih banyak yang dapat dikembangkan dari program tersebut, seperti halnya pencarian *query* menerima masukan lebih dari 2 kata. Akhir kata, penulis berharap makalah ini dapat bermanfaat khususnya bagi penulis dan umumnya bagi semua pihak yang berkepentingan.

Link Video Youtube

Berikut tautan youtube untuk melihat video dari penjelasan dan demonstrasi dari tugas makalah penulis.

https://youtu.be/NqviFSVmvgE

Ucapan Terima Kasih

Puji syukur penulis panjatkan kepada Allah SWT yang telah memberikan rahmat serta karunianya sehingga pada akhirnya, penulis dapat menyelesaikan makalah ini tepat pada waktunya. Penulis menyadari bahwa tanpa adanya bantuan dan dorongan dari berbagai pihak, penyelesaian makalah ini tidak akan terwujud. Oleh karena itu, dengan ketulusan dan kerendahan hati, penulis mengucapkan terima kasih dan penghargaan setinggi-tingginya kepada Dr. Nur Ulfa Maulidevi, S.T., M.Sc. sebagai dosen pengampu mata kuliah IF2211 Strategi Algoritma kelas 02 yang telah mendidik penulis selama satu semester. Tidak lupa, penulis juga mengucapkan terima kasih kepada keluarga dan teman-teman penulis yang telah memberikan dukungan yang luar biasa kepada penulis.

Referensi

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf, diakses pada 20 Mei 2022

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf, diakses pada 20 Mei 2022

[3] https://www.w3schools.com/python/python_regex.asp, diakses pada 20 Mei 2022

[4] Utami, Sri. (2018). Kuliner Sebagai Identitas Budaya: Perspektif Komunikasi Lintas Budaya.

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, 20 Mei 2022

Eiffel Aqila Amarendra (13520074)