LAPORAN TUGAS
TEORI BAHASA DAN OTOMATA
"Mesin Pencari Dengan Finite Otomata"

Disusun Oleh :

Kelompok 1

I Putu Ryan Paramaditya	**(1808561024)**
I Gede Aditya Mahardika Pratama	**(1808561028)**
I Kadek Gowinda	**(1808561033)**
Lalu Muhamad Waisul Kuroni	**(1808561037)**
Edo krishnanda Aditya	**(1808561042)**

Kelas : B

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS UDAYANA

BADUNG

2020

KATA PENGANTAR

Puji dan syukur ke hadirat Tuhan Yang Maha Kuasa atas segala rahmat yang diberikan-Nya sehingga tugas Laporan Tugas Teori Bahasa dan Otomata yang berjudul ”*Mesin Pencari Dengan Finite Otomata*” ini dapat kami selesaikan. Laporan ini kami buat sebagai kewajiban untuk memenuhi tugas mata perkuliahan Tugas Teori Bahasa. Dalam kesempatan ini, kami ingin menghaturkan terimakasih yang dalam kepada semua pihak yang telah membantu menyumbangkan ide dan pikiran mereka demi terwujudnya makalah ini. Akhirnya saran dan kritik pembaca yang dimaksud untuk mewujudkan kesempurnaan laporan ini, penulis sangat hargai.

Badung, 17 November 2020

Tim Penulis

DAFTAR ISI

BAB I

PENDAHULUAN

1.1 Latar Belakang

Mesin Finite Otomata merupakan dasar dari mesin pencari text (*text searching*). Dengan memecah Bahasa menjadi bagian-bagian kecil penyusunnya, sebuah mesin dapat diciptakan menggunaka algoritma yang sederhana.

Ada dua jenis finite otomata yaitu ; Deterministic Finite Automata (DFA) dan Non-Deterministic Finite Automata (NFA). Dalam membuat mesin pencari kata, akan diperlukan kedua jenis finite otomata. Pertama input akan dibuatkan bentuk dari NFA yang selanjutnya akan dikonversi ke dalam bentuk DFA kemudian pencarian kata dalam dokumen akan dilakukan.

Mempelajari serta mengimplementasikan mesin otomata sederhana akan menjadi langkah awal untuk menuju *text mining*. Mengerti cara kerja mesin pencari text akan menjadi dasar dari *text mining* itu sendiri. Hal ini tentunya akan sangat bermanfaat bagi programmer yang baru belajar dan ingin mengembangkan minatnya terhadap penambangan tekstual.

1.2 Tujuan

1.2.1 Mengimplementasikan NFA dan DFA untuk membuat mesin pencari.

1.2.2 Menciptakan mesin pencari berbasis *finite automata*.

1.3 Manfaat

1.3.1 Dapat mengimplementasikan NFA dan DFA kedalam mesin pencari.

1.3.2 Dapat mengimplementasikan mesin pencari dengan *web programming*.

1.3.3 Memahami dasar mesin pencari dengan mengerti penggunaan mesin otomata.

BAB II

KAJIAN PUSTAKA

2.1 Finite Automata

Dalam teori automata, jika setiap transisi secara unik ditentukan oleh status sumber dan simbol input, dan setiap transisi status perlu membaca simbol input, mesin status hingga disebut deterministic finite automata (DFA). nondeterministic finite automata (NFA) atau nondeterministic finite-state machine tidak perlu mematuhi batasan ini. Secara khusus, setiap DFA juga merupakan NFA. Terkadang, istilah NFA digunakan dalam rentang yang lebih sempit dan merujuk pada NFA yang bukan DFA.

Dengan menggunakan algoritma konstruksi subset, setiap NFA dapat diubah menjadi DFA yang setara; yaitu, DFA dapat mengenali bahasa formal yang sama. Seperti DFA, NFA hanya mengenali bahasa alami. NFA diajukan oleh Michael O. Rabin dan Dana Scott pada tahun 1959. Mereka juga membuktikan bahwa NFA dan DFA adalah setara. NFA digunakan untuk implementasi ekspresi reguler: Konstruksi Thompson adalah algoritme yang digunakan untuk membuat ekspresi reguler menjadi NFA, yang dapat secara efektif melakukan pencocokan pola pada string. Sebaliknya, algoritma Kleene dapat digunakan untuk mengubah NFA menjadi ekspresi reguler (biasanya eksponensial dalam input otomatis).

2.2 Bahasa Pemrograman Python

Python adalah bahasa pemrograman yang ditafsirkan multiguna. Tidak seperti bahasa lain yang sulit dibaca dan dipahami, python lebih menekankan pada keterbacaan kode agar lebih mudah dalam memahami sintaksnya. Bagi para pemula dan yang sudah mahir dalam bahasa pemrograman lain, hal ini membuat Python sangat mudah dipelajari. Bahasa pertama kali muncul pada tahun 1991 dan dirancang oleh seorang pria bernama Guido van Rossum. Sejauh ini, Python Software Foundation masih mengembangkan Python. Bahasa Python mendukung hampir semua sistem operasi, bahkan untuk sistem operasi Linux, hampir semua distro menyertakan Python. Dengan menggunakan kode yang sederhana dan

mudah, programmer dapat memprioritaskan pengembangan aplikasi yang sedang dikembangkan.

Django adalah sebuah framework full-stack untuk membuat aplikasi web dengan bahasa pemrograman python. Django juga merupakan framework python web tingkat tinggi yang dapat melakukan pengembangan aplikasi dengan cepat dan memiliki desain pragmatis yang bersih.Django dapat membuat pengembangan aplikasi menjadi lebih mudah, lebih cepat dan lebih sedikit menggunakan kode.

Kerangka kerja Django memiliki template, pustaka dan API yang dirancang untuk tumbuh dan terhubung bersama. Django cocok untuk semua proyek, dari proyek kecil hingga proyek besar. Django menggunakan Python, yang merupakan salah satu bahasa pemrograman paling populer di tahun 2015. Sekarang, merupakan bahasa pemrograman yang paling banyak dipelajari. Dibandingkan dengan framework lainnya, Django adalah yang paling kaya fitur. Django berisi semua yang dibutuhkan untuk membangun aplikasi.

2.3 Sistem Pencarian Teks

Konsep yang digunakan pada proyek ini yaitu menemukan string dalam teks seperti Mesin pencari menggunakan teknologi tertentu, yang disebut indeks terbalik, di mana untuk setiap kata yang muncul di Web (ada 100.000.000 kata yang berbeda) daftar semua tempat di mana kata itu muncul disimpan. Teknik indeks terbalik tidak menggunakan otomata terbatas, tetapi juga membutuhkan waktu yang sangat lama bagi crawler untuk menyalin Web dan menyiapkan indeks. Ada sejumlah aplikasi terkait yang tidak cocok untuk indeks terbalik, tetapi merupakan aplikasi yang bagus untuk teknik berbasis otomata.

Penggunaan Non-deterministic Finite Automata (NFA) pada pencarian teks, misalkan kita diberi sekumpulan kata, yang akan kita sebut kata kunci, dan kita ingin menemukan kemunculan kata-kata tersebut. Dalam aplikasi seperti ini, cara yang berguna untuk melanjutkan adalah merancang robot terbatas nondeterministik, yang memberi sinyal, dengan memasuki keadaan menerima, bahwa ia telah melihat salah satu kata kunci. Teks dokumen dimasukkan, satu karakter pada satu waktu ke NFA ini, yang kemudian mengenali kemunculan kata

kunci dalam teks ini. Ada bentuk sederhana untuk NFA yang mengenali sekumpulan kata kunci.

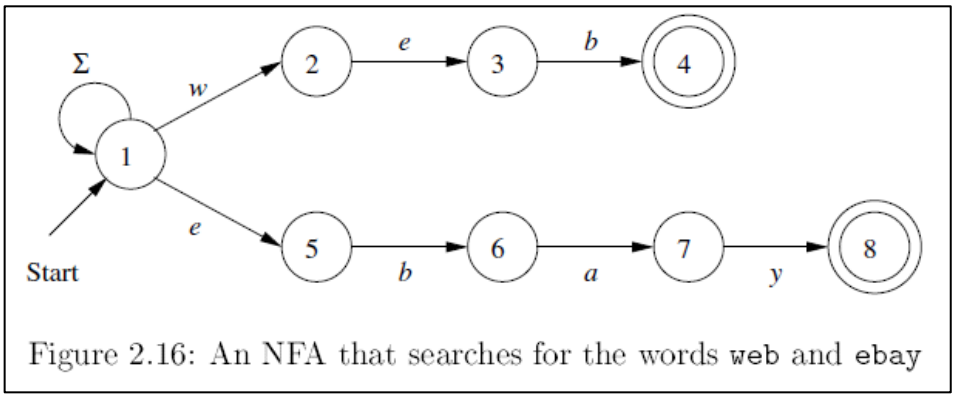


Figure 2.16: An NFA that searches for the words `web` and `ebay`

Contoh diatas bagaimana proses NFA pada pencarian teks yaitu mengenali kata “web” dan “ebay”. State 1 adalah state awal, kami menggunakan sigma untuk mewakili himpunan semua karakter ASCII yang dapat dicetak. State 2 hingga 4 mengenali kata “web” , dan state 5 hingga 8 mengenali kata “ebay”

BAB III

PEMBAHASAN

3.1 Sistem Search Engine

System yang kami bangun adalah system Search Engine yang merupakan sebuah system yang mempunyai fungsi utama untuk membantu pengguna dalam mencari dokumen atau berkas yang tersedia, sehingga dapat mempermudah dan mempercepat pengguna untuk mencari informasi. sistem Search Engine tersebut dibangun dengan arsitektur berbasis website.

Alur pada sistem search engine adalah pengguna melakukan proses input keyword pada search bar, selanjutnya sistem akan melakuakan pengecekan pada document atau berkas yang telah tersimpan dalam sistem. Setelah proses pengecekak selesai, sistem akan menampilkan time consumed per iterasi, jumlah document yang mengandung keyword, daftar document dan snippet yang mengandung minimal sebuh keyword yan dicari sesuai iterasi. Selain itu user dapat melihat isi document hasil pencarian dan juga user dapat melihat Quintuple NFA yang berdasarkan keyword.

Tampilan atau front end pada system search engine tersebut kami menggunakan CSS, framework Django, dan framework bootstrap. Sedangkan pada bagian back end atau system inti kami menggunakan Bahasa pemrograman Python. Serta pada proses pengcodingan system kami lakukan menggunakan code editor Visual Studio Code.

3.2 Pengimplementasian Sistem

3.2.1 Pencarian Dokumen

Proses pencarian pada system memiliki dua buah sekenario yakni, sekenario pertama setelah user memberikan keyword, system akan melakukan pengecekan document dari doc 1-50, doc 1-100, doc 1-150, dan seterusnya yang bertambah secara bertahap. Sedangkan scenario kedua setelah user memberikan keyword, sistem akan melakukan pengecekan dokumen dari doc 1-50, 51-100, 101-150, dan seterusnya bergantian secara bertahap.

```
def delta_topi(text, state, symbol, delta, start_state, final_state):
    stop = False
    state_hasil = [start_state]
    for i, char in enumerate(text):
        # print(str(i)+' '+char)
        state_cek = state_hasil
        state_hasil = [start_state]
        for state in state_cek:
            if char in delta[state]:
                if state is start_state:
                    if any(item in delta[state][char] for item in final_state):
                        key = final_state[''.join(
                            [j for j in delta[state][char] if j in final_state])]
                        if check_word(text, i, key):
                            index = i
                            stop = True
                            break
                    state_hasil = list(
                        set(delta[state][char]) | set(state_hasil))
                else:
                    if delta[state][char] in final_state:
                        key = final_state[delta[state][char]]
                        if check_word(text, i, key):
                            index = i
                            stop = True
                            break
                    if delta[state][char] not in state_hasil:
                        state_hasil.append(delta[state][char])
        if stop:
            break

    # KONDISI APAKAH DIDALAM TEXT TERDAPAT SALAH SATU DARI KEYWORD
    if stop:
        return str(index)
    else:
        return False
```

Pada sistem ini proses pembentukan quintuple dengan cara setelah user menginputkan keyword, akan dilanjutkan dengan pembentukan mesin automata dengan menggunakan NFA. Pembentukan NFA dijalankan pada function *generate(keyword).* Proses yang pertama dilakukan adalah mengubah string kyeword menjadi sebuah list, lalu element list yang mengandung empty string akan dihapus. Ketika list tersebut tidak memiliki element atau dalam artian user tidak menginputkan string, maka program akan memberikan nilai error. Namun jika tidak, akan dilanjutkan pada proses selanjutnya yaitu pembentukan quintuple dari NFA.

Himpunan *State* ditentukan berdasarkan banyaknya karakter dari keyword, himpunan *Symbol* ditentukan berdasarkan karakter unik pada keyword, *Start state* ditentukan langsung oleh program. Lalu untuk pembentukan transisi dari tiap *state* akan dilakukan perulangan dari setiap karakter per keryword, dimana transisi akan dimulai dari *start state* lalu dilanjutkan pada penentuan transisi dari state selanjutnya berdasarkan perulangan dari tiap karakter keyword. Ketika perulangan sampai pada karakter terakhir dari tiap keyword maka state hasil transisi dari pembacaan karakter tersebut akan disimpan pada list *final state.*

```
def create_NFA(keyword):
    # INIT QUINTUPLE
    state = []
    symbol = []
    delta = {}
    start_state = 'S0'
    final_state = {}
    # MANY OF STATE
    n = 1
    for i in keyword:
        n += len(i)
    # CREATE STATE
    for i in range(n):
        state.append('S'+str(i))
        delta['S'+str(i)] = {}
    # CREATE SYMBOL
    symbol = list(set(''.join(map(str, keyword))))
    # Create DELTA
    delta_NFA(keyword, delta, final_state)
    return state, symbol, delta, start_state, final_state
```

Setelah quintuple NFA didapatkan, dilanjutkan dengan pemanggilan function *detla_topi()*. Dimana function tersebut digunakan untuk menegecek apakah text dari suatu dokumen mengandung salah satu keyword yang diinputkan oleh user. Proses dimulai dengan melakukan perulangan tiap karakter dari text dokumen. Tiap iterasi akan melakukan pengecekan terhadap transisi dari tiap karakter. Ketika suatu state tidak dapat membaca karakter tersebut maka program akan mengasumsikan bahwa hasil transisi tersebut menuju ke start state.

Namun ketika suatu state dapat membaca karakter tersebut maka akan dilakukan pengecekan lagi, apakah hasil transisi merupakan final state. Jika tidak akan dilanjutkan pada pengecekan transisi selanjutnya. Jika iya, maka akan dicek kata yang mengandung keyword tersebut, apakah sebelum atau sesudah kata tersebut terdapat alphabet atau angka. Jika iya, maka kata tersebut tidak valid atau kata tersebut bukanlah kata yang dicari oleh user. Namun jika tidak, program akan berhenti melakukan perulangan dan memberikan tanda bahwa text tersebut mengandung salah satu dari keyword.

Proses pemilihan document yang akan menjadi hasil dari sistem dilakukan dengan cara membaca file secara terurut mengikuti scenario yang telah disiapkan dan diseleksi seusai dengan hasil perhitungan delta topi yang dilakukan oleh sistem, jika document memenuhi syarat maka dokument tersebut akan tersimpan sebagai hasil dan nantinya akan ditampilkan pada sistem serta setiap keyword yang termuat dalam dokument tersebut akan di cetak bold pada tampilan hasil pencarian. Pada sistem pencarian juga terdapat fitur untuk mengganti jumlah dokument. Fitur ini berfungsi untuk mengatur jumlah dokument yang akan diseleksi oleh sistem.

Code untuk melakukan perulangan untuk pencarian dokument :

```
for x in range(int(jumlah[0])):
    f = open(list_text[x], 'r', errors='ignore')
    content = f.read()
    f.close()
    data = delta_topi(content.lower(), state, symbol, delt
                      start_state, final_state)
    if data:
        doc_name = list_text[x][5:-4]
        text_res = content[int(data)-len(keys)+1:]
        text_res = change_text(
            text_res, keys.split(" "))
        text_res = mark_safe(text_res)
        result.append({'doc_name': doc_name, 'res': text_r
jumlah = jumlah[0]
```

Tampilan untuk search bar dan pengaturan jumlah dokument :

3.2.2 Melihat Isi Dokumen

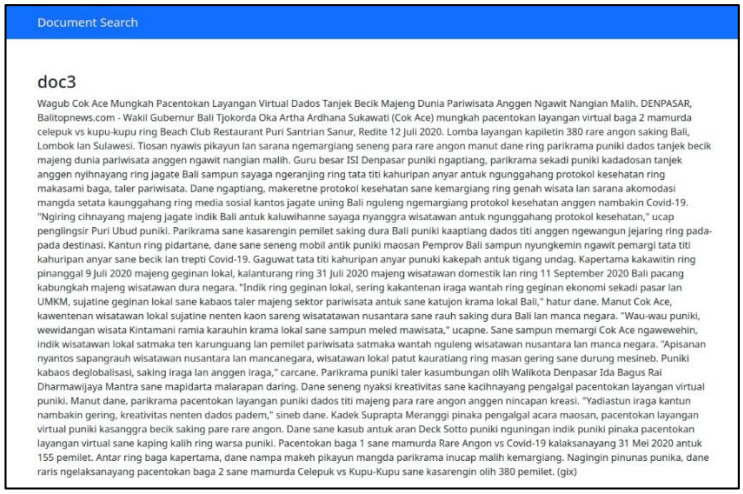
Sistem memiliki fitur untuk melihat dokument hasil secara keseluruhan atau lebih detail dengan cara melakukan load pada file yang dipilih dan menampilkannya ,

Code untuk melihat dokument :

```
def detail(request, doc_name):

    f = open("docs/"+doc_name+".txt", 'r', errors='ignore')
    content = f.read()
    f.close()
    context = {'title': doc_name, 'data': content}
    return render(request, 'search/detail.html', context)
```

Tampilan pad saat melihat dokument secara lebih jelas :



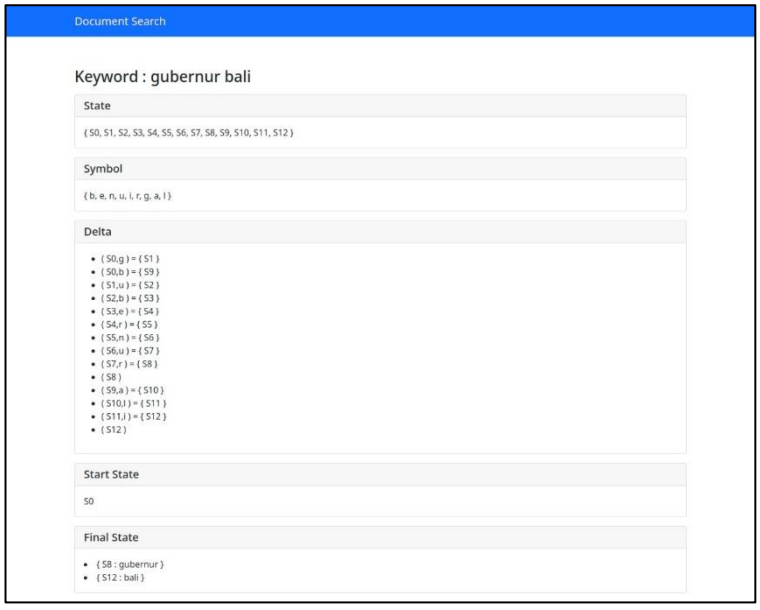
3.2.3 Melihat Hasil Quintuple

Sistem memiliki fitur untuk melihat hasil quintuple dari keyword yang telah di berikan user. Quintuple akan diperlihatkan dalam began-bagan yang menunjukkan tiap elemen yang terdapat pada mesin finite otomata.

Code untuk menampilkan quintuple :

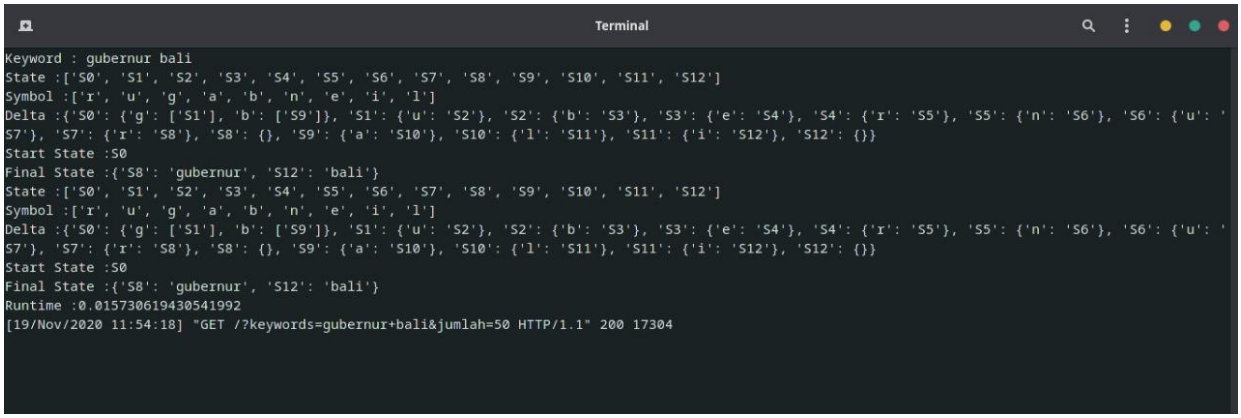
```
def quintuple(request):
    keys = request.GET['keyword']
    state, symbol, delta, start_state, final_state = main(keys)
    context = {'key': keys, 'state': state, 'symbol': symbol, 'delta': delta,
               'start_state': start_state, 'final_state': final_state}
    return render(request, 'search/quintuple.html', context)
```

Tampilan fitur quintuple :

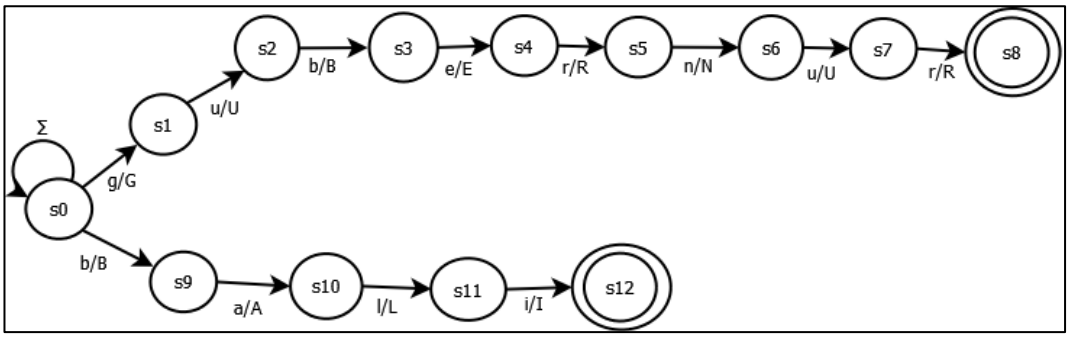


3.3 Percobaan dan Hasil

3.3.1 Evaluasi Blackbox



Sistem dapat menerima lebih dari satu kata inputan dengan spasi sebagai pemisah. Percobaan menggunakan kata kunci “gubernur bali” akan menghasilkan sebuah mesin NFA yang akan dikonversi menjadi DFA. Sehingga mesin akan terlihat seperti diagram dibawah ini.



Dari hasil Blackbox, mesin memiliki 13 state, dan 2 buah final state yaitu {s8,s12}. Hasil yang ditunjukkan sama dengan diagram menyatakan jika mesin berjalan dengan baik

3.3.2 Evaluasi Dengan Perubahan Dokumen

. Dalam Sistem, kita dapat merubah jumlah dokumen atau dokumen mana yang akan digunakan dengan rentang 50 dokumen. Adapun hasil pencarian dengan waktu yang dibutuhkan dapat dilihat dibawah ini.

Hasil pencarian jika kita petakan kedalam table.

Jumlah Dokumen	Hasil Yang Didapat	Estimasi Waktu (ms)
50	45	0.02
100	91	0.20
150	137	0.66
200	179	0.74

Hasil ini menunjukkan bagaimana jumlah dokumen akan mempengaruhi waktu yang dibutuhkan dalam mendapatkan hasil. Semakin banyak jumlah dokumen yang dicari, maka semakin banyak waktu yang dibutuhka untuk mendapatkan hasil.

Hasil pencarian berdasarkan dokumen dengan rentang 50 dokumen.

Dokumen Ke-	Hasil Yang Didapat	Estimasi Waktu (ms)
0-50	45	0.01
50-100	46	0.01
100-150	46	0.02

Berdasarkan perubahan dokumen, hasil menunjukkan tidak ada perubahan yang signifikan dari waktu yang diperlukan. Namun tetap ada perbedaan waktu pencarian walaupun hanya 0.01ms.

Dari kedua alur yang kita gunakan, dapat diketahui jika jumlah dokumen dapat mempengaruhi waktu eksekusi yang dibutuhkan mesin. Selain itu dokumen yang berbeda dengan jumlah yang sama juga akan memberikan sedikit perbedaan waktu karena jumlah dokumen yang mengandung kata kunci kemungkinan berbeda, serta isi dari dokumen juga kemungkinan berbeda.

BAB IV

PENUTUP

4.1. Simpulan

Sistem mesin NFA dan DFA yang digunakan dapat berjalan dengan baik untuk integrasi kedalam sistem website. Sistem dapat mencari 45 dari 50 dukomen dalam waktu kurang dari 0.02ms dan mencari 179 dari 200 dokumen dalam waktu 0.74ms.

Dengan keyword yang sama, jumlah dokumen yang dimasukkan tidak mempengaruhi akurasi, namun membebani sedikit waktu tambahan yang diperlukan. Dengan jumlah dokumen yang sama namun masukan dokumen yang berbeda, juga akan mempengaruhi lamanya waktu pencarian, karena tiap dokumen memiliki panjang yang berbeda serta tempat keyword pada tiap dokumen yang juga berbeda.

4.2. Saran

Dalam project kali ini, penulis menemukan sedikit kendala dalam menentukan bagaimana kriteria dokumen yang harus ditampilkan. Hal ini terjadi karea tidak meratanya format dari isi dokumen. Ada beberapa dari dokumen yang hanya berisi link saja dan ada beberapa yang tidak memiliki judul. Inilah alasan kami menampilkan dokumen dengan nama filenya (doc01, doc02,….dst).

Kedepannya akan sangat baik jika dokumen yang ada lebih teratur. Dengan memberikan format yang sama, agar pencarian dokumen dapat dilakukan dengan lebih terstruktur.

DAFTAR PUSTAKA

Django documentation | Django (no date). Available at: https://docs.djangoproject.com/en/3.1/ (Accessed: 16 November 2020).

Mishra, K. and Chandrasekaran, N., 2008. *Theory Of Computer Science*. 3rd ed. New Delhi: Prentice Hall.

Lutz, M., 2013. *Learning Python*. 5th ed. Canada: O'Reilly Media, Inc.

Link GitHub:: https://github.com/Labenea/DocumentSearch