**Penerapan String Matching dan Regular Expression dalam**

**Pembangunan Deadline Reminder Assistant**

**LAPORAN TUGAS BESAR**

Diajukan Untuk Memenuhi Tugas IF2211 Strategi Algoritma

Semester II Tahun 2020/2021

Disusun oleh

**Muhammad Azhar Faturahman (13519020)**

**Reihan Andhika Putra (13519043)**

**Hokki Suwanda (13519143)**

**TEKNIK INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

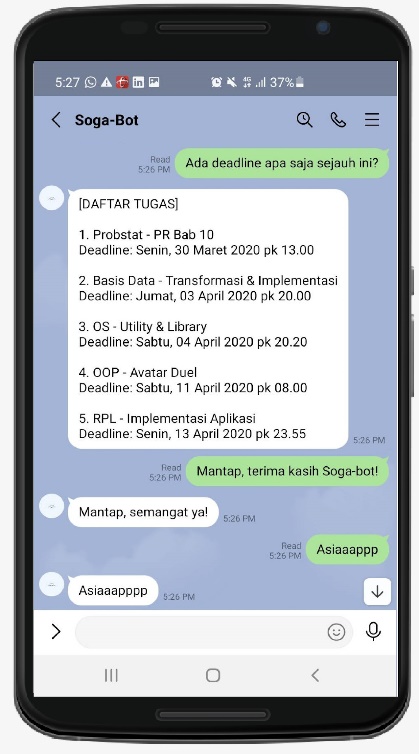
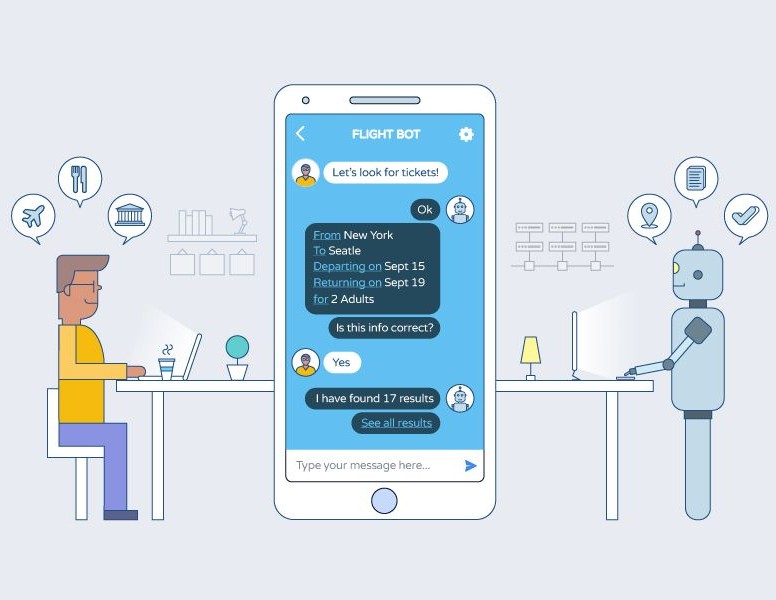
**BANDUNG**

**2020**

**BAB I**

**DESKRIPSI TUGAS**

Bukan sesuatu yang janggal lagi jika semakin hari tugas-tugas di Teknik Informatika Semester 4 semakin bertambah banyak. Hal ini tentunya berakibat pada bertambahnya kegiatan dan pekerjaan yang harus dilakukan mahasiswa. Tak jarang pula ada tugas yang terlupakan karena mahasiswa sulit untuk mengingat semua tugas dan deadline tersebut. Oleh karena itu, mahasiswa Teknik Informatika berniat untuk membuat suatu *Google Assistant* sederhana berupa Deadline Reminder Assistant, atau dalam bahasa Indonesia adalah Asisten Pengingat Deadline.



*Gambar 1. Ilustrasi Chatbot dan Asisten Pengingat Deadline*

Sumber : <https://id.pinterest.com/pin/824299538024636729/>dan dokumentasi pribadi

Di era digital ini, kita tentu sudah pernah mendengar teknologi atau aplikasi seperti Chatbot, LINE Bot, atau Google Assistant. Ketiganya merupakan agen cerdas yang meniru kemampuan manusia untuk melakukan percakapan dengan *user*. Kehadiran Chatbot ini tentu membantu kehidupan manusia, khususnya dalam membantu menyajikan informasi yang diperlukan user dan menjawab berbagai pertanyaan yang sering ditanyakan oleh *user.* Secara spesifik dalam konteks Asisten Pengingat Deadline ini, Chatbot tersebut akan menjawab pertanyaan-pertanyaan mahasiswa yang sering ditanyakan seperti deadline seminggu ke depan, deadline di bulan ini, dan task-task penting lainnya yang perlu dilakukan. Chatbot ini akan sangat membantu user agar tidak lagi melewatkan deadline tugas.

# Deskripsi tugas:

Dalam tugas besar ini, Anda akan diminta untuk membangun sebuah *chatbot* sederhana yang berfungsi untuk membantu mengingat berbagai deadline, tanggal penting, dan task-task tertentu kepada user yang menggunakannya. Dengan memanfaatkan algoritma String Matching dan Regular Expression, Anda dapat membangun sebuah *chatbot* interaktif sederhana layaknya Google Assistant yang akan menjawab segala pertanyaan Anda terkait informasi deadline tugas-tugas yang ada.

# Fitur-Fitur Aplikasi:

Deadline Reminder Assistant. akan dibangun dengan sistem **Question and Answer** dimana pengembang diharapkan sudah menyediakan kumpulan formula tertentu untuk melakukan pendeteksian setiap perbedaan command atau perintah pada aplikasi Chatbot. Berikut ini adalah runtutan fitur yang dimiliki oleh Deadline Reminder Assistant tersebut.

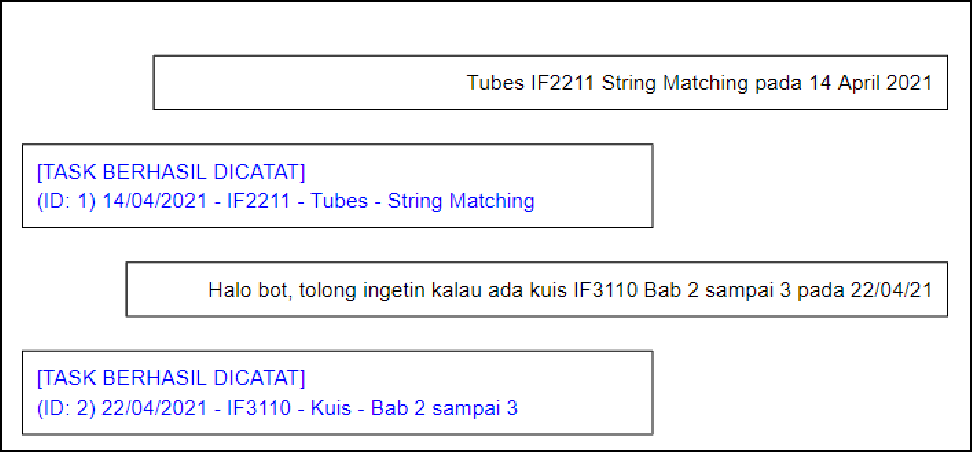
1. Menambahkan *task* baru
   1. Suatu kalimat *diklasifikasikan* sebagai suatu *task* apabila mengandung **semua komponen** berikut ini:
2. Tanggal (format dibebaskan)
3. Kode Mata Kuliah / Nama Mata Kuliah (dibebaskan)
4. Jenis Tugas (berdasarkan daftar **kata penting** yang sudah disediakan)
5. Topik Tugas (tidak ada batasan)
   1. Point 1 sampai dengan 4 diklasifikasikan menggunakan **regular expression**

sehingga masukan kalimat benar-benar layaknya kalimat sehari-hari

* 1. Jika pesan berhasil dikenali oleh assistant, maka assistant akan mengirim pesan balasan yang berisi ID (sesuai urutan task diinput), tanggal, kode mata kuliah, jenis tugas, dan topik tugas. Contoh pesan balasan dari bot sebagai berikut.

[TASK BERHASIL DICATAT]

(ID: 1) 14/04/2021 - IF2211 - Tubes - String matching

* 1. Contoh interaksi

1. Melihat daftar *task* yang harus dikerjakan=
   1. Seluruh *task* yang sudah tercatat oleh assistant

* Contoh perintah yang dapat digunakan: “Apa saja deadline yang dimiliki sejauh ini?”
  1. Berdasarkan periode waktu
     1. Pada periode tertentu (DATE\_1 until DATE\_2)

Contoh perintah yang dapat digunakan: “Apa saja deadline antara DATE\_1 sampai DATE\_2?”

* + 1. N minggu ke depan

Contoh perintah yang dapat digunakan: “Deadline N minggu ke depan apa saja?”

* + 1. N hari ke depan

Contoh perintah yang dapat digunakan: “Deadline N hari ke depan apa saja?”

* + 1. Hari ini

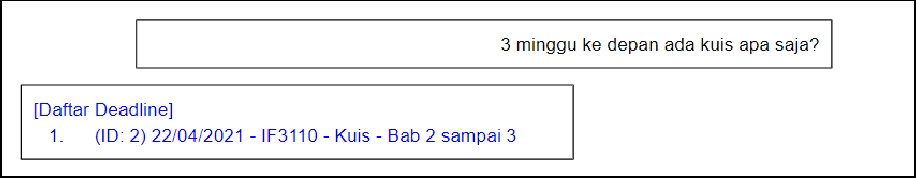
Contoh perintah yang dapat digunakan: “Apa saja deadline hari ini?”

* 1. Berdasarkan jenis task (kata penting)
     1. Sesuai dengan daftar task yang didefinisikan
     2. User dapat melihat daftar task dengan jenis task tertentu
     3. Misalnya: “3 minggu ke depan ada kuis apa saja?”, maka Chatbot akan menampilkan daftar kuis selama 3 minggu kedepan

# Catatan:

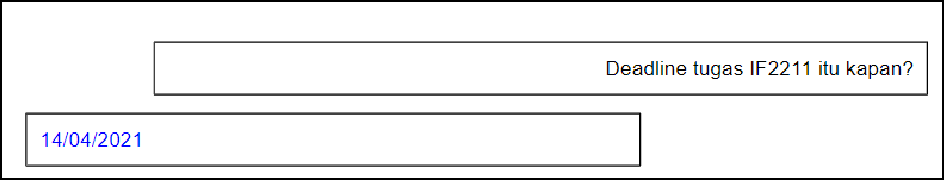
Eksekusi perintah pengguna bisa mencakup ketiga poin sekaligus sehingga formula pengenalan command sebaiknya dibuat sebagai satu kesatuan utuh.

Contoh interaksi



# Keterangan penting:

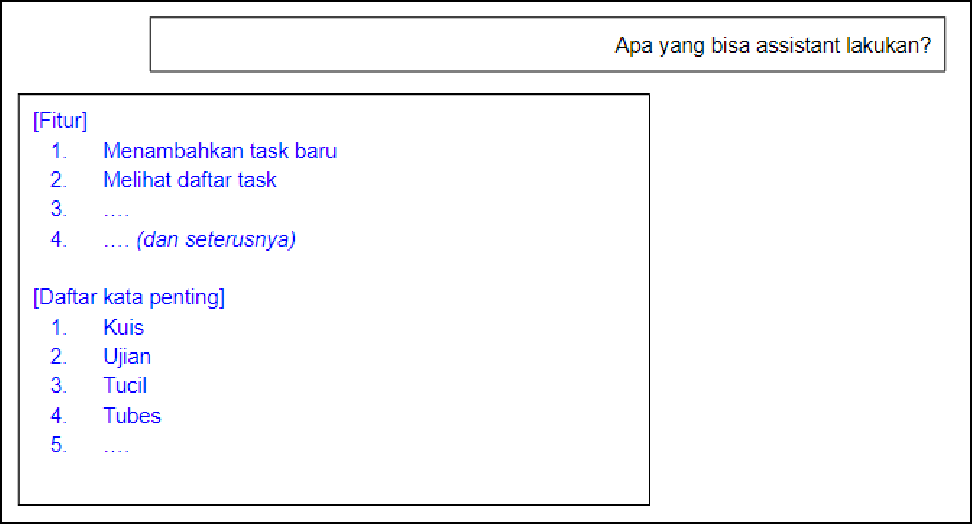
* + - * Perintah yang digunakan pengguna bisa tidak selalu sama, asalkan mengandung kata kunci yang ditentukan (kata kunci tiap perintah bisa ditentukan sendiri). Misal kedua contoh di bawah ini memberikan output yang sama
        + Apa saja **deadline** antara **03/04/2021** sampai **15/04/2021**?
        + Antara **03/04/2021** dan **1 5/04/2021** ada **deadline** apa saja ya?

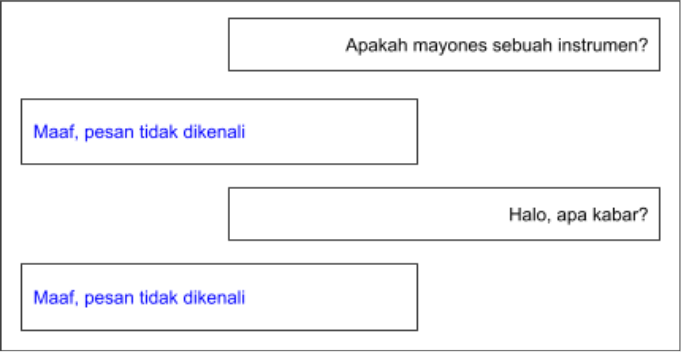
1. Menampilkan deadline dari suatu task tertentu
   1. Hanya berlaku untuk task yang bersifat **Tugas** atau memiliki tenggat waktu
   2. Misalnya: “Deadline tugas IF2211 itu kapan?”
   3. Contoh interaksi
2. Memperbaharui task tertentu
   1. Memperbarui **tanggal** dari suatu task (dalam kehidupan nyata, tentu ada kejadian dimana deadline dari suatu task diundur)
   2. Perintah yang dimasukkan meliputi 1 keyword untuk memperbaharui suatu task dan nomor task tertentu.
   3. Misalnya:

* “Deadline task X diundur menjadi 28/04/2021” dimana X merupakan nomor ID dari suatu task.
  1. Apabila task berhasil diperbaharui, Chatbot akan menampilkan pesan sukses memperbaharui suatu task. Sebaliknya, Chatbot akan menampilkan pesan error apabila task yang dimaksud tidak dikenali oleh Chatbot (belum masuk ke dalam Daftar Task)

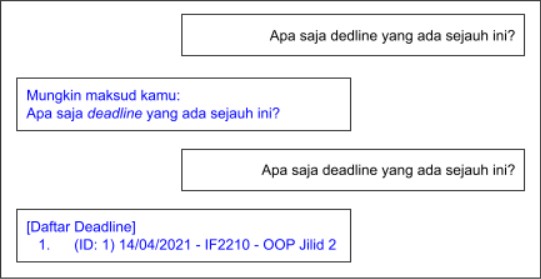
1. Menandai bahwa suatu task sudah selesai dikerjakan
   1. Apabila user sudah menyelesaikan suatu task, maka task tersebut bisa ditandai bahwa task tersebut sudah selesai dan tidak perlu lagi ditampilkan pada Daftar Task selanjutnya.
   2. Misalnya:

* “Saya sudah selesai mengerjakan task X” dimana X merupakan nomor ID dari suatu task.
  1. Apabila perintah yang dimasukkan user bisa dieksekusi, Chatbot akan menampilkan pesan sukses. Sebaliknya, Chatbot akan menampilkan pesan error apabila task yang dimaksud tidak dikenali oleh Chatbot (belum masuk ke dalam Daftar Task)

1. Menampilkan opsi *help* yang difasilitasi oleh assistant
   1. Berisikan command-command yang dapat digunakan oleh user
   2. Misalnya: “Apa yang bisa assistant lakukan?”
   3. Bot akan memberikan hasil berupa daftar kata-kata yang bisa digunakan untuk menambahkan dan melihat daftar task (setiap kelompok bebas membentuknya seperti apa)
   4. Contoh interaksi
2. Mendefinisikan list kata penting terkait apakah itu merupakan suatu *task* atau tidak
   1. Minimal terdapat 5 kata penting berbeda, contohnya adalah: [“Kuis”, “Ujian”, “Tucil”, “Tubes”, “Praktikum”]
   2. Kata penting akan digunakan pada penentuan jenis tugas dari suatu *task*.
   3. Daftar kata penting tidak perlu dibuat dinamis, cukup static saja atau hardcoded.
3. Menampilkan pesan error jika assistant tidak dapat mengenali masukan user.
   1. Masukan yang tidak termasuk ke dalam jenis pesan di poin 1 sampai 4 dapat dikategorikan sebagai masukan tak dikenali.
   2. Error message dibebaskan sesuai kreativitas mahasiswa
   3. Contoh interaksi



1. **(Bonus)** Chatbot dapat memberikan rekomendasi kata jika terdapat kesalahan kata (*typo*) pada perintah yang ditulis pengguna
   1. Berikan rekomendasi kata jika perintah masukan pengguna *mismatch* dengan daftar kata yang diterima chatbot, namun masih memiliki tingkat kemiripan di atas 75%.
   2. Contoh interaksi



* 1. Ada berbagai metriks yang dapat dimanfaatkan untuk mencari kemiripan kata, salah satunya adalah Levenshtein distance yang diukur melalui pendekatan *dynamic programming*. Anda dapat mempelajari Levenshtein distance melalui pranala [ini](https://medium.com/%40ethannam/understanding-the-levenshtein-distance-equation-for-beginners-c4285a5604f0).

**BAB II**

**LANDASAN TEORI**

1. **Algoritma KMP**

Algoritma Knuth-Morris-Pratt atau yang biasa disebut sebagai KMP adalah sebuah algoritma yang digunakan untuk mencari sebuah *pattern* pada text dari kanan ke kiri, algoritma ini mirip dengan algorita *brute force*. Perbedaan antara algoritma ini dengan *brute force* adalah algoritma ini akan melakukan penggeseran terhadap pattern secara lebih pintar daripada secara *brute force*.

Cara penggeseran yang dilakukan oleh algoritma KMP adalah dengan cara menggeser *string* *pattern* ke *substring* yang memiliki suffix yang sama dengan prefix terpanjang pada *pattern* yang merupakan sebuah suffix pada *pattern* tersebut. Contoh : pattern adalah “abaab” prefix “abaab” yang merupakan suffix “abaab” maka yang dipilih adalah “ab” dan jumlah pergeseran yang bisa dilaksanakan sekaligus adalah panjang *pattern* dikurangi panjang string yang terpilih yaitu,5- 2 = 3.

Cara kerja KMP disebut sebagai *preprocessing*, karena program telah mengetahui lokasi kesamaan prefix dan suffix sehingga perpindahan bisa langsung ke daerah tersebut, dan jika pada *string* dimana suffix tidak terdapat pada prefix maka *pattern* akan selalu melakukan pergeseran terhadap *string* teks. Kompleksitas waktu dari algoritma KMP adalah O(m+n), dibandingkan dengan *brute force* KMP merupakan algoritma yang sangat cepat. Berikut adalah implementasi algoritma KMP dengan menggunakan bahasa python

**def** **kmpMatch**(text, pattern, notCaseSensitive=True):

"""Mencari kemunculan pertama pattern pada text

Args:

text (String): string yang akan dicari kemunculan pattern

pattern (String): pattern suatu string

caseSensitive (Boolean) : caseSensitive pattern terhadap string, default False

Returns:

Integer: Index pertama kemunculan pattern, -1 jika tidak ditemukan

"""

**if** notCaseSensitive:

text = text.lower()

pattern = pattern.lower()

fail = computeFail(pattern)

n = len(text)

m = len(pattern)

j = **0**

i = **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**

**def** **computeFail**(pattern):

"""Mencari prefix terbesar dari suffix pattern

Args:

pattern (String): Pattern yang ingin dicari

Returns:

Array of Integer: Array yang menyatakan prefix terbesar dari suatu suffix pattern pada posisi ke i

"""

fail = [**0** **for** i **in** range(len(pattern))]

m = len(pattern)

j = **0**

i = **1**

**while** i < m:

**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

Kelebihan dari algoritma KMP adalah algoritma ini tidak pernah perlu bergerak kebelakang pada input text. Hal ini membuat algoritma ini bagus dalam memproses file yang sangat besar yang dibaca dari media eksternal. Kelemahan KMP adalah dengan bertambahnya alfabet pada string, KMP akan semakin *error* dan ketidakcocokan akan lebih sering ditemukan. Biasanya ketidakcocokan terjadi di awal dari string, tetapi jika ketidakcocokan ditemukan di bagian akhir dari string maka KMP akan bergerak sangat cepat. KMP dasar tidak melihat apa yang menyebabkan ketidakcocokan sehingga tidak dapat langsung digeser setelah ketidakcocokan.

1. **Algoritma BooyerMoore**

Algoritma Boyer-Moore algoritma pencocokan string yang didasari oleh dua teknik, yaitu teknik *the looking glass* dan teknik *the character jump*. Teknik *the looking glass* merupakan teknik pencocokan *pattern* pada string dimulai dari indeks terakhir *pattern* dan indeks pada string dimulai dari awal menyesuaikan dengan indeks *pattern*. Sedangkan teknik *the character jump* merupakan teknik melompat jika terjadi *mismatch* pada karakter *pattern* dan *string*.

Pada algoritma Boyer-Moore terdapat tiga kasus yang digunakan untuk teknik the *character jump*, yakni untuk kasus pertama jika pada *pattern* mengandung karakter X yang merupakan karakter *mismatch* pada string, maka geser pattern tersebut agar sejajar dengan karakter yang *mismatch* dengan *string*. Kasus kedua adalah jika *pattern* mengandung karakter X yang merupakan karakter *mismatch* pada string namun tidak mungkin menyejajarkan keduanya, maka *pattern* digeser sebanyak satu. Kasus ketiga terjadi ketika *pattern* tidak mengandung karakter yang terjadi *mismatch* pada string, maka langsung lompati *pattern* hingga melewati karakter yang menjadi *mismatch* tersebut.

Pada algoritma Boyer-Moore untuk menentukan seberapa banyak lompatan karakter yang dilakukan apabila terjadi *mismatch*, digunakan fungsi pendukung berupa *last occurence*. Fungsi *last occurence* ini dimiliki oleh tiap karakter berbeda pada *string*, kemudian menentukan indeks terakhir ditemukannya karakter pada *pattern*. Jika tidak ditemukan karakter *string* pada *pattern* maka nilai *last occurence* untuk karakter tersebut adalah -1.

Berikut merupakan implementasi algoritma Boyer-Moore dalam bahasa python

**def** **bmMatch**(text, pattern, notCaseSensitive=True):

"""Mencari kemunculan pertama pattern pada text

Args:

text (String): string yang akan dicari kemunculan pattern

pattern (String): pattern suatu string

caseSensitive (Boolean) : caseSensitive pattern terhadap string, default False

Returns:

Integer: Index pertama kemunculan pattern, -1 jika tidak ditemukan

"""

**if** notCaseSensitive:

text = text.lower()

pattern = pattern.lower()

last = buildLast(pattern)

n = len(text)

m = len(pattern)

i = m-**1** # Index pencarian di text

j = m-**1** # Index pencarian di pattern

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

# Tidak ada hasil jika pattern lebih panjang daripada text

**return** -**1**

**while** (i < n):

**if** pattern[j] == text[i]:

**if** j == **0**:

**return** i

**else**:

i -= **1**

j -= **1**

**else**:

lo = last[ord(text[i])] # Kemunculan terakhir character text pada pattern

i = i + m - min(j, **1**+lo)

j = m-**1**

# Gagal ditemukan

**return** -**1**

**def** **buildLast**(pattern):

"""Menghasilkan array yang merepresentasikan index kemunculan terakhir karakter

Args:

pattern (String): Pattern suatu strin

Returns:

Array of Integer: array yang merepresentasikan index kemunculan terakhir karakter

"""

last = [-**1** **for** i **in** range(**128**)]

**for** i **in** range(len(pattern)):

last[ord(pattern[i])] = i

**return** last

1. **Regex**

*Regular Expression* (Regex) adalah sebuah notasi yang dapat digunakan untuk mendeskripsikan pola dari kata yang ingin dicari. *Regular expression* juga salah satu metode dalam pencarian data dalam string. *Regular expression* tidak seperti algoritma pencocokan string pada umumnya karena didalam *regular expression* terdapat dua tipe karakter yaitu *literal character* dan *metacharacter*. *Literal character* ialah karakter biasa yang benar-benar ada wujudnya misalnya seperti yang terdapat pada ASCII yang terdiri dari berbagai huruf, angka, hingga tanda baca. Lalu yang membuat regular expression ini lebih unggul dari algoritma pencocokan string lainnya yaitu adanya *metacharacter*. *Metacharacter* adalah karakter yang tidak memiliki wujud seperti *literal character* namun memiliki aturan yang bisa menunjukan apakah suatu string dapat diterima atau tidak.

*Metacharacter* bisa dibilang seperti pola atau ciri sebagai acuan dalam pencarian string. *Metacharacter* pada regular expression menghasilkan *finite state machine (FSM*) untuk mengolah string masukan. FSM adalah alur dalam memeriksa masukan berdasarkan kondisi status yang ada. FSM memiliki luaran diterima atau tidak. Masukan diterima apabila kondisi status terakhir sesuai dengan status akhir FSM. Sintaks dari Regex berupa kumpulan karakter khusus yang digunakan untuk menentukan pola pencarian pada teks. Sintaks Regex memiliki beberapa aturan seperti berikut.

1. **Character Class**

Aturan ini membentuk character class dalam konstruksi regex.

|  |  |
| --- | --- |
| **Construct** | **Deskripsi** |
| [abc] | a, b, atau c (simple class) |
| [^abc] | Semua karakter **selain** a,b,c (negasi) |
| [a-zA-Z] | a sampai z atau A sampai Z, inclusive (range) |
| [a-d[m-p]] | a sampai d atau m sampai p (gabungan) |
| [a-z&&[def]] | d, e atau f (irisan) |
| [a-z&&[^bc]] | a sampai z, kecuali b dan c (substraksi) |
| [a-z&&[^m-p]] | a sampai z, dan bukan m sampai p (substraksi) |

1. **Predefined Character Classes**

Predefined class membuat Regex lebih mudah dibaca dan mengurangi kesalahan.

|  |  |
| --- | --- |
| **Construct** | **Deskripsi** |
| . | Semua karakter |
| \d | Digit [0-9] |
| \D | Non digit [^0-9] |
| \s | Whitespace character [\t\n\x0B\f\r] |
| \S | Non whitespace character [^\s] |
| \w | Word character [a-zA-Z\_0-9] |
| \W | Non word character [^\w] |

1. **Quantifier**

Quantifier digunakan untuk mendefinisikan jumlah perulangan pola

|  |  |
| --- | --- |
| **Construct** | **Deskripsi** |
| x? | x muncul **satu** atau tidak sama sekali |
| x\* | x muncul **nol** atau banyak |
| x+ | x muncul satu atau banyak |
| x{n} | x muncul tepat n kali |
| x{n,} | x muncul setidaknya n kali |
| x{n,m} | x muncul antara n sampai m kali |

1. **Boundary Matcher**

Boundary matcher digunakan untuk mencari pola yang muncul di posisi tertentu.

Misalnya di awal atau di akhir.

|  |  |
| --- | --- |
| **Construct** | **Deskripsi** |
| ^ | Awal baris |
| $ | Akhir baris |
| \b | Batas kata |
| \B | Batas bukan kata |
| \G | Akhir match sebelumnya |
| \Z | Akhir dari input tapi untuk final terminator jika ada |
| \z | Akhir dari input |

1. **Chatbot**

*Chatbot* merupakan salah satu program dalam kecerdasan buatan yangdirancang untuk dapat berkomunikasi langsung dengan manusia sebagaipenggunanya. Yang membedakan *chatbot* dengan sistem pemrosesan bahasaalami (*Natural Language Processing System*) adalah kesederhanaan algoritmayang di gunakan. Meskipun banyak *bot* yang dapat menginterpretasikan danmenanggapi input manusia, sebenarnya bots tersebut hanya mengartikan katakunci dalam input dan membalasnya dengan kata kunci yang paling cocok, ataupola kata-kata yang paling mirip dari data yang telah ada dalam database yangtelah dibuat sebelumnya.

*Chatbot* terdiri dari dua komponen utama yakni *bot program* dan *brain**file*. *Bot program* merupakan program utama pada *chat bot* yang akan mengakses*input* dari pengguna, melakukan *parsing* dan kemudian membawanya ke *brain file**(knowledge base)* untuk kemudian diberikan respon. *Brain file* merupakan otak dari *chat bot* itu sendiri yang menentukanbagaimana cara *chat bot* berpikir dan akan memberikan respon. *Brain file*berfungsi sebagaimana tabel informasi (*knowledge base*). Di dalam *brain file*inilah disimpan semua kosakata, kepribadian, dan pengetahuan (*knowledge*) dari*chatbot*. Semakin banyak pengetahuan yang dimiliki *chat bot* maka akan semakin besar ukuran file dari *brain file* tersebut. Adapun *bot program* sendiriterdiri dari komponen *scanner* dan *parser.*

*Scanner* merupakan salah satu bagian dari kompilator bahasa pada komputer yang bertugas melakukan analisis leksikal. *Scanner* menerima *input* berupa *stream* karakter kemudian memilah program sumber menjadi satuan leksik yang disebut dengan *token*. *Token* ini akan menjadi *input* bagi *parser*.

*Parser* atau *syntactic analyzer* pada kompilator bahasa pemrograman berfungsi untuk memeriksa kebenaran kemunculan setiap *token*. Pada *Chatbot* *system*, fungsi dari *parser* ini agak berbeda karena *token* yang akan diolah, semuanya memiliki tipe yang sama yaitu berupa kata (*word*). Urutan kemunculan *token* akan diolah dengan mengacu pada *brain file* agar didapatkan makna kalimat yang sesungguhnya. Dengan kata lain, tahap analisa semantik terjadi di bagian *brain file*. Kemampuan dari *parser* untuk mengolah *token* dan bekerja sama dengan *brain file* inilah yang paling menentukan tingkat kecerdasan dari sebuah *chatbot*.

*Bot program* atau bagian aplikasi menentukan kemampuan dan keterampilan *chat bot* untuk berbicara pada anda atau pada pengguna lainnya, atau dengan kata lain *bot program* berperan sebagai mulut. *Chatbot* tidak tahu apa-apa, tidak punya nama dan tidak punya kepribadian. Untuk itu pengguna harus mengajarinya berbagai hal sehingga ia dapat berbicara dengan baik dan semua

pelajaran tersebut dimasukkan ke *brain file*

**BAB III**

**Analisis Pemecahan Masalah**

1. **Langkah Penyelesaian Masalah Setiap Fitur**
2. **Fitur Fungsionalitas dan Arsitektur Chatbot Yang Dibangun**

**BAB IV**

**IMPLEMENTASI DAN PENGUJIAN**

1. **Spesifikasi Teknis Program**
2. **Penjelasan Tata Cara Penggunaan Program**
3. **Hasil Pengujian**
4. **Analisis Hasil Pengujian**

**BAB V**

**KESIMPULAN DAN SARAN**

1. **Kesimpulan**

Dari tugas besar IF 2211 Strategi Algoritma semester 2 2020/2021 berjudul “Penerapan String Matching dan Regular Expression dalam Pembangunan Deadline Reminder Assistant” kami berhasil membuat sebuah *chatbot* sederhana dengan memanfaatkan algoritma KMP/Boyer-Moore dan Regex. *Chatbot* ini dapat menerima input berupa permintaan *user* untuk menambah *task*, menampilkan *task* yang sudah ada, menampilkan *deadline*, mengubah *deadline* suatu task, dan menyelesaikan suatu *task.* Pengecekan *input* tidak dilakukan secara *exact matching* namun input dibebaskan kepada *user* selama mengandung *keyword* tertentu.

1. **Saran**

Saran-saran yang dapat kami berikan untuk tugas besar IF 2211 Strategi Algoritma semester 2 2020/2021 adalah:

* 1. Algoritma yang *dig*unakan pada Tugas Besar ini masih memiliki banyak kekurangan sehingga sangat memungkinkan untuk dilakukan efisiensi, misalnya dengan tidak menggunakan fungsi yang sama berulang-ulang. Oleh karena itu, dalam pengembangan program ini, masih bisa dilakukan efisiensi kinerja.
  2. Memperjelas spesifikasi dan batasan-batasan setiap program pada *file* tugas besar untuk mencegah adanya multitafsir dan kesalahpahaman pada proses pembuatan program.

1. **Refleksi**

Setelah menyelesaikan tugas besar IF 2211 Strategi Algoritma semester 2 2020/2021, kami dapat merefleksikan beberapa hal, yaitu:

1. Komunikasi antar anggota kelompok berjalan dengan baik, sehingga tidak terjadi miskomunikasi atau kesalahpahaman selama pengerjaan.
2. Selama pengerjaan Tugas Besar, telah dibuat beberapa milestone untuk setiap orangnya dengan deadline yang bervariasi dan semua terselesaikan tepat waktu.
3. Selama proses pembuatan program, ketika ditemukan ketidaksesuaian saat proses eksperimen, temuan langsung dikomunikasikan kepada anggota kelompok lainnya dan bersama-sama mencari solusi.
4. Perlunya untuk mempelajari Web development agar kedepannya dapat membuat sebuah web app dengan UI yang lebih fungsional dan lebih user-friendly.
5. Lebih merapikan source code program karena ada beberapa fungsi yang didefinisikan secara tidak modular.
6. Mengerjakan tugas besar dengan perasaan gembira, karena it’s not worth it if you’re not have fun.
7. **Komentar**

Setelah menyelesaikan tugas besar IF 2211 Strategi Algoritma semester 2 2020/2021, kami mempunyai beberapa komentar, yaitu:

1. Yey akhirnya tubes stima kelar :D

**DAFTAR PUSTAKA**

Efendi, I. (2016, December 18). Pengertian ALGORITMA Greedy. <https://www.it-jurnal.com/pengertian-algoritma-greedy/> (diakses tanggal 14 Februari 2021).

(Materi tentang Greedy).

EntelectChallenge, B. (2019, September 02). Entelectchallenge/2019-worms. <https://github.com/EntelectChallenge/2019-Worms> (diakses tanggal 14 Februari 2021).

(Entellect Github)

Apa itu game engine. (2012, March 08). <https://rickykurn.wordpress.com/2012/03/08/apa-itu-game-engine/> (diakses tanggal 14 Februari 2021).

(Game Engine)

.