Aplikasi Algoritma *Divide and Conquer* dan Pencocokan *String* dalam Pembagian Prioritas Kebutuhan dalam Spesifikasi Tugas Besar

Owen Christian Wijaya - 13520124
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13520124@std.stei.itb.ac.id

***Abstract*—Dalam pelaksanaannya, tugas besar mempunyai esensi berupa implementasi materi di kelas, akan tetapi mahasiswa diminta untuk mengimplementasikan dalam bentuk program yang belum pernah diajarkan sebelumnya. Oleh karena itu, mahasiswa terkadang kewalahan karena tidak hanya harus membagi waktu dalam implementasi materi kelas, namun juga untuk implementasi bentuk program dalam spesifikasi tugas besar. Dengan menggunakan algoritma *divide and conquer* dan pencocokan *string*, daftar kebutuhan-kebutuhan dalam sebuah spesifikasi tugas besar dapat diurutkan sesuai dengan pembobotan yang dihitung berdasarkan kata-kata kunci dalam kebutuhan**

Keywords—algoritma; divide and conquer; pencocokan string; pembagian prioritas

I. PENDAHULUAN

Tugas besar, atau lebih akrab disebut dengan "tubes" di kalangan mahasiswa Teknik Informatika Institut Teknologi Bandung, adalah sebuah kegiatan sebagai bagian dari kurikulum hampir setiap mata kuliah yang ada di Teknik Informatika. Sebuah tugas besar mempunyai kebutuhan-kebutuhan yang banyak untuk dipenuhi, dan oleh karena itu, biasanya dikerjakan secara berkelompok (mulai dari tiga hingga enam orang). Kebutuhan-kebutuhan ini biasanya dicantumkan dalam sebuah dokumen spesifikasi, yang dibagikan secara daring oleh tim asisten sebuah mata kuliah menjelang pengumuman sebuah tugas besar.

Gambar 1. Spesifikasi Tugas Besar III IF2211 Strategi Algoritma Semester 2 Tahun 2020/2021

Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Tugas-Besar-3-IF2211-Strategi-Algoritma-2022.pdf. Diakses 3 Mei 2022.

Kebutuhan-kebutuhan yang tercantum dalam sebuah dokumen spesifikasi terbagi menjadi dua jenis secara garis besar. Terdapat kebutuhan yang bersangkutan dengan materi yang diajarkan di dalam kelas, dan kebutuhan yang bersangkutan dengan implementasi program. Implementasi-implementasi ini biasanya berbeda-beda per tugas besarnya. Selain itu, kebutuhan ini tidak digabungkan per kelompok kebutuhan, sehingga terkadang mahasiswa kesulitan untuk menentukan bagian mana yang seharusnya dikerjakan terlebih dahulu dan bagian mana yang lebih penting.

Sisi positifnya adalah mahasiswa diberikan kebebeasan untuk melakukan eksplorasi terhadap bahasa-bahasa pemrograman dan teknologi-teknologi yang ada dalam implementasi secara mandiri. Sisi negatifnya adalah tidak semua mahasiswa mempunyai pengetahuan yang setara tentang sebuah bahasa pemrograman yang diperlukan untuk mengerjakan sebuah tugas besar tersebut. Selain itu, perbedaan tingkat kemampuan per mahasiswa dan kesibukan per mahsiswa akan menentukan banyaknya waktu eksplorasi yang dimiliki

Melalui mata kuliah IF2211 Strategi Algoritma, penulis merasa bahwa masalah ini dapat diatasi menggunakan algoritma *divide and conquer*. Dengan membagi kebutuhan-kebutuhan yang tercantum dalam spesifikasi, seorang mahasiswa dapat membuat *roadmap* atau peta perencanaan pengerjaan terurut berdasarkan prioritas sebuah kebutuhan. Selain itu, pengerjaan tugas besar dapat dibagi antar mahasiswa apabila mengetahui klasifikasi dari sebuah kebutuhan. Oleh karena itu, diharapkan algoritma *divide and conquer* dapat membantu mahasiswa dalam merencanakan pengerjaan sebuah tugas besar.

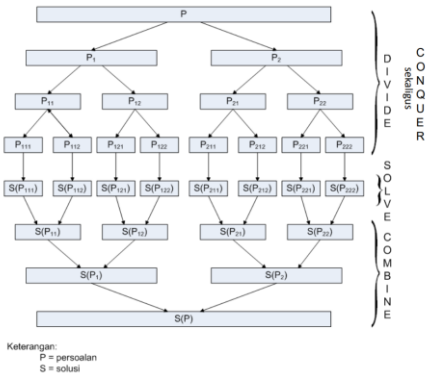
II. TEORI DASAR

A. Algoritma Divide and Conquer

Algoritma *divide and conquer* (lebih lanjut direferensikan di dokumen ini sebagai algoritma DNC) adalah sebuah algoritma yang sering digunakan untuk menyelesaikan permasalahan-permasalahan yang ada di bidang ilmu komputer. Algoritma ini mempunyai keunikan dibandingkan

algoritma-algoritma lainnya, di mana algoritma ini menyelesaikan sebuah permasalahan dalam tiga langkah:

- Langkah *divide* : Membagi persoalan yang ada menjadi beberapa upa-persoalan yang mempunyai kemiripan dengan persoalan awalnya, namun berukuran lebih kecil. Idealnya, tiap upa-persoalan mempunyai ukuran yang sama. Pembagian sebuah persoalan menjadi upa-persoalan akan dilakukan secara rekursif selama ukuran upa-persoalan belum mencapai ukuran yang diinginkan. Proses akan berlangsung secara rekurens hingga ukuran upa-persoalan mencapai ukuran yang diinginkan. Misalnya, dalam penyelesaian *merge sort* sebuah larik linear tidak terurut, sebuah larik akan dibagi terus menerus hingga ukuran tiap upalarik sebesar dua elemen.
- Langkah *conquer* / *solve* : Penyelesaian masing-masing upa-persoalan yang telah dibentuk. Penyelesaian dapat dilakukan langsung secara prosedural apabila ukuran upa-persoalan sudah cukup untuk melakukan penyelesaian langsung. Apabila ukuran upa-persoalan cukup besar, penyelesaian dapat dilakukan secara rekursif apabila memungkinkan.
- Langkah *combine*: Pada tahap combine, solusi-solusi dari upa-persoalan digabungkan kembali menjadi sebuah kesatuan yang merepresentasikan solusi dari permasalahan awal. Pada penyelesaian merge sort, misalnya, tiap solusi upa-larik akan digabungkan hingga membentuk larik terurut.



Gambar 2. Visualisasi Algoritma *Divide And Conquer*

Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf. Diakses 4 Mei 2022.

Mayoritas aplikasi algoritma *divide and conquer* bersifat rekursif; pembagian terus-menerus hingga didapatkan upapersoalan dengan ukuran yang diinginkan. Oleh karena itu, untuk menghitung kompleksitas waktu sebuah algoritma *divide and conquer*, digunakan sebuah teorema bernama Teorema Master. Teorema ini dapat digunakan untuk menentukan notasi asimptotik kompleksitas waktu yang berbentuk relasi rekurens. Misalnya terdapat sebuah relasi rekurens seperti berikut:

$$T(n) = a\,T(n / b) + cn^d \text{ dengan persyaratan:}$$

$$n = b^k,\ k > 0,\ a \geq 1,\ b \geq 2,\ c \text{ dan } d \geq 0,$$

maka kompeksitas waktu T(n) adalah:

- $O(n^d)$ apabila $a < b^d$
- $O(n^d \log(n))$ apabila $a = b^d$
- $O(n^{\log_b a})$ apabila $a > b^d$

B. *Pencocokan String*

Pencocokan *string* adalah jenis algoritma yang digunakan dalam pengembangan perangkat lunak untuk menemukan kemunculan sebuah pola (*pattern*) yang diinginkan dalam sebuah teks (*text*). Pencocokan *string* diterapkan dalam banyak fitur-fitur dalam perangkat lunak yang kita pakai sehari-hari, seperti:

- Fitur pencarian kata dalam editor teks untuk menemukan kata yang diinginkan dan mengganti sebuah kata dengan kata lain (*find and replace*)
- Pencocokan *query* masukan dalam *search engine* seperti Google untuk menemukan situs yang diinginkan
- Pencocokan rantai DNA untuk menentukan penyakit tertentu

Ada tiga buah algoritma pencocokan *string* yang umum digunakan:

1. Algoritma *Brute Force*

Algoritma *brute force* melakukan pencocokan dengan menggeser *pattern* per satu karakter apabila tidak ditemukan kecocokan. Algoritma ini melakukan pcncocokan secara naif dengan mencocokkan satu-per-satu karakter dalam *pattern* dengan karakter dalam *text*. Apabila kecocokan tidak ditemukan, *pattern* akan digeser 1 karakter ke kanan, dan pencocokan diulangi lagi.

2. Algoritma Knuth-Morris-Pratt

Algoritma Knuth-Morris-Pratt (KMP) adalah algoritma pencocokan *string* yang didasari oleh algoritma *brute force* dengan heuristik tambahan. Algoritma KMP melakukan pencocokan dengan mencocokkan satu-per-satu karakter dalam *pattern* dengan karakter dalam *text*. Apabila kecocokan tidak ditemukan, *pattern* akan digeser sebanyak jumlah awalan yang cocok dengan akhiran *pattern* tersebut untuk *subtext* yang sedang dicocokkan. Perhitungan penggeseran untuk *pattern* dilakukan pada *border function* yang menghasilkan larik berisi jumlah penggeseran yang dilakukan untuk kecocokan pada indeks ke-i dari *pattern*.

3. Algoritma Boyer-Moore

Algoritma Boyer-Moore adalah algoritma yang melakukan pencocokan *string* dari kanan-ke-kiri (*looking glass technique*), tidak seperti dua algoritma sebelumnya yang melakukan pencocokan dari kiri-ke-kanan. Algoritma ini melakukan pemrosesan terhadap *pattern* dengan menghitung indeks terakhir kemunculan sebuah karakter K pada *pattern*. Indeks ini kemudian akan disimpan dalam sebuah larik yang akan dijadikan patokan dalam pergeseran.

Pergeseran dalam algoritma Boyer-Moore menggunakan teknik *character-jump* yang dilakukan saat terjadi

ketidakcocokan pada teks T[i] = karakter K atau ketidakcocokan antara *pattern* dan *text*. Teknik ini mencakup tiga buah kasus:

- Apabila *pattern* P mengandung karakter K, geser *pattern* ke kanan untuk mensejajarkan kemunculan terakhir dari karakter K di dalam teks dengan karakter di teks T[i]
- Apabila *pattern* P mengandung karakter K, namun tidak bisa digeser ke kanan, maka geser P sebanyak 1 karakter ke teks T[i + 1]
- Apabila kedua kasus tidak ditemukan, geser *pattern* P untuk mensejajarkan P[0] dengan teks T[i + 1]

Pada implementasi nanti, pencocokan *string* akan dilakukan untuk mencocokkan kata-kata di dalam masukan dengan kata-kata kunci yang disediakan. Setiap masukan pada program akan dihitung bobotnya dengan akumulasi bobot-bobot dari tiap kata yang ada dalam masukan. (lebih lanjut dibahas di Bab IV: Implementasi). Karena pencocokan dilakukan pada kata-kata secara sederhana, penulis menilai bahwa algoritma *brute force* cocok untuk digunakan karena pencocokan yang tidak memakan waktu yang panjang dengan ukuran teks dan *pattern* yang relatif sama (tidak mempunyai selisih jauh).

C. Tugas Besar

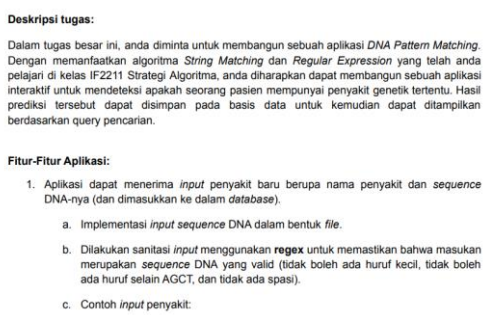
Tugas besar adalah salah satu bagian dari kurikulum banyak mata kuliah di Institut Teknologi Bandung. Tugas besar termasuk salah satu hal yang dianggap mempunyai prioritas tinggi bagi mayoritas mahasiswa karena tugas besar tidak hanya mempunyai persentase yang tinggi di nilai akhir, tetapi juga memberikan kesempatan untuk bereksplorasi dan implementasi materi-materi yang diajarkan di kelas. Selain itu, sebuah tugas besar mempunyai banyak kebutuhan-kebutuhan yang harus diimplementasikan. Setiap kebutuhan yang wajib atau diperbolehkan untuk diimplementasikan biasanya dituliskan dalam sebuah dokumen spesifikasi yang dibagikan secara daring oleh tim asisten sebuah mata kuliah bersangkutan menjelang atau setelah pengumuman sebuah tugas besar.

Secara garis besar, terdapat dua jenis kebutuhan yang biasanya harus diimplementasikan:

- Kebutuhan mata kuliah, yaitu kebutuhan yang bersangkutan dengan materi-materi yang diajarkan pada mata kuliah tersebut. Karena tugas besar adalah rangka implementasi materi yang diajarkan di saat kelas, maka biasanya intisari dari sebuah tugas besar adalah untuk melakukan implementasi dari materi yang diajarkan. Intisari ini berbeda-beda antara mata kuliah dan tergantung dengan waktu tugas besar tersebut diberikan.

- Kebutuhan program, yaitu kebutuhan yang bersangkutan dengan implementasi program yang diinginkan. Kebutuhan kedua adalah kebutuhan yang bersangkutan dengan implementasi program. Kebutuhan ini bersangkutan dengan bagaimana sebuah program ingin diimplementasikan, dalam wadah apa sebuah program ingin diimplementasikan, dan teknologi-teknologi apa saja yang diinginkan dalam implementasi tersebut. Implementasi-implementasi ini biasanya berbeda-beda per tugas besarnya. Contohnya adalah tugas besar pada mata kuliah IF221 Strategi Algoritma pada semester genap tahun ajaran 2021/2022:

- Tugas besar pertama mata kuliah IF2211 Strategi Algoritma mengimplementasikan bot permainan Overdrive pada bahasa pemrograman Java dan cukup dijalankan di command-line interface
- Tugas besar kedua mata kuliah IF2211 Strategi Algoritma mengimplementasikan desktop app pencarian folder dalam bahasa C#
- Tugas besar ketiga mata kuliah IF2211 Strategi Algoritma mengimplementasikan website pencocokan DNA dengan kakas JavaScript dan server menggunakan kakas Node.js/Golang



Deskripsi tugas:

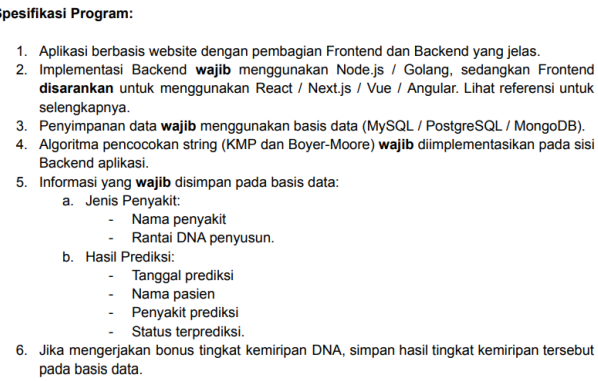
Dalam tugas besar ini, anda diminta untuk membangun sebuah aplikasi *DNA Pattern Matching*. Dengan memanfaatkan algoritma *String Matching* dan *Regular Expression* yang telah anda pelajari di kelas IF2211 Strategi Algoritma, anda diharapkan dapat membangun sebuah aplikasi interaktif untuk mendeteksi apakah seorang pasien mempunyai penyakit genetik tertentu. Hasil prediksi tersebut dapat disimpan pada basis data untuk kemudian dapat ditampilkan berdasarkan query pencarian.

Fitur-Fitur Aplikasi:

1. Aplikasi dapat menerima *input* penyakit baru berupa nama penyakit dan *sequence* DNA-nya (dan dimasukkan ke dalam *database*).
   a. Implementasi *input sequence* DNA dalam bentuk *file*.
   b. Dilakukan sanitasi *input* menggunakan **regex** untuk memastikan bahwa masukan merupakan *sequence* DNA yang valid (tidak boleh ada huruf kecil, tidak boleh ada huruf selain AGCT, dan tidak ada spasi).
   c. Contoh *input* penyakit:

Gambar 3. Contoh Kebutuhan Mata Kuliah

Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Tugas-Besar-3-IF2211-Strategi-Algoritma-2022.pdf. Diakses 6 Mei 2022.



Spesifikasi Program:

1. Aplikasi berbasis website dengan pembagian Frontend dan Backend yang jelas.
2. Implementasi Backend **wajib** menggunakan Node.js / Golang, sedangkan Frontend **disarankan** untuk menggunakan React / Next.js / Vue / Angular. Lihat referensi untuk selengkapnya.
3. Penyimpanan data **wajib** menggunakan basis data (MySQL / PostgreSQL / MongoDB).
4. Algoritma pencocokan string (KMP dan Boyer-Moore) **wajib** diimplementasikan pada sisi Backend aplikasi.
5. Informasi yang **wajib** disimpan pada basis data:
   a. Jenis Penyakit:
      - Nama penyakit
      - Rantai DNA penyusun.
   b. Hasil Prediksi:
      - Tanggal prediksi
      - Nama pasien
      - Penyakit prediksi
      - Status terprediksi.
6. Jika mengerjakan bonus tingkat kemiripan DNA, simpan hasil tingkat kemiripan tersebut pada basis data.

Gambar 4. Contoh Kebutuhan Program

Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Tugas-Besar-3-IF2211-Strategi-Algoritma-2022.pdf. Diakses 6 Mei 2022.

Selain itu, biasanya ada kebutuhan-kebutuhan bonus yang memberikan tambahan nilai bagi mahasiswa yang menerapkan. Kebutuhan bonus ini bersifat tidak wajib, dan biasa dikerjakan apabila mahasiswa telah menyelesaikan kebutuhan wajib dan masih mempunyai waktu lebih untuk mengerjakan bonus, atau ingin melakukan eksplorasi lebih lanjut.

Sisi positif dari adanya kebutuhan demikian adalah seperti berikut:

- Mahasiswa tidak hanya berkesempatan untuk menerapkan yang dipelajari di kelas, namun juga melakukan eksplorasi terhadap teknologi-teknologi baru atau bahasa pemrograman baru yang sebelumnya belum diajarkan di kelas. Hal ini menjadi bekal mahasiswa untuk melakukan kegiatan-kegiatan tugas besar di semester-semester selanjutnya.

- Mahasiswa dapat membagi jatah pekerjaan pada teman-teman sekelompoknya. Apabila ada teman sekelompok yang mempunyai pengalaman lebih pada implementasi program, pekerjaan dapat dibagi dengan sama rata dan tugas besar dapat dikerjakan bersama.

- Mata kuliah yang teoritis memberikan kesempatan implementasi dunia-nyata bagi para mahasiswa dengan *platform* yang berbeda-beda, sehingga mahasiswa mempunyai pengalaman dalam mengembangkan program di *platform* berbeda-beda.

Akan tetapi, pengerjaan tugas besar seperti ini juga akan memberikan dampak negatif, contohnya adalah:

- Tidak semua mahasiswa dapat melakukan eksplorasi secara bebas apabila ada banyak tugas besar lain dari mata kuliah lainnya yang harus dikerjakan karena prioritas mahasiswa terbagi ke beberapa tugas besar. Selain itu, perbedaan kemampuan tiap mahasiswa dapat memberikan *peer pressure* pada mahasiswa-mahasiswa lainnya.

- Banyaknya fitur yang diberikan dapat membuat seorang mahasiswa bingung harus memulai dari mana. Selain itu, kebutuhan-kebutuhan yang dituliskan tidak mempunyai klasifikasi tertentu tentang urutan pengerjaan atau tingkat kesulitan, sehingga mahasiswa harus menentukan urutan pengerjaan sendirian secara relatif. Akibatnya, mahasiswa kewalahan dalam menentukan pembagian pengerjaan tugas besar, sehingga terkadang merasa terbeban untuk mengerjakan semuanya sendirian atau tidak berpartisipasi sama sekali dan menyerahkan semuanya ke kelompok.

- Spesifikasi yang diberikan tidak diurutkan berdasarkan prioritas atau tidak. Tidak ada pembobotan yang jelas terhadap urgensi sebuah kebutuhan yang diberikan, sehingga pengkategorian urgensi sulit untuk dilakukan.

  Salah satu metode yang sederhana dan cepat untuk melakukan pembagian pengerjaan sebuah tugas besar dalam suatu kelompok adalah dengan melihat kata-kata kunci (*keywords*) yang terlampir. Secara umum, kebutuhan-kebutuhan dapat diurutkan kepentingannya dengan melihat kata-kata kunci dalam sebuah kalimat. Kebutuhan-kebutuhan dengan kata-kata "*frontend*", "*backend*", atau "visualisasi" cenderung dianggap sebagai kebutuhan yang harus terselesaikan terlebih dahulu, sedangkan kebutuhan-kebutuhan dengan kata "komentar", "bugtesting", atau "laporan" biasanya diprioritaskan menjelang pengumpulan karena sifatnya yang tidak terlalu mendesak dan dapat dilakukan setelah pengerjaan program selesai.

III. Penentuan Pembagian Pekerjaan Tugas Besar menggunakan Algoritma Divide and Conquer

Menggunakan algoritma *divide and conquer*, kita dapat menentukan urutan pekerjaan yang harus dilakukan dalam suatu tugas besar. Persoalan yang dibahas adalah sebagai berikut: diberikan sebuah daftar kebutuhan dalam sebuah tugas besar tanpa ada urutan pembobotan atau kategori pengerjaan tugas.

Daftar kebutuhan yang diberikan berupa deskripsi singkat mengenai intisari dari sebuah kebutuhan yang diberikan yang dapat disertakan dengan sebuah kata sifat yang menandakan urgensi sebuah kebutuhan. Kata-kata yang dimasukkan akan dikenali oleh program dan diberikan bobot tergantung urgensi dan kata-kata kunci lainnya. Contoh masukan program adalah sebagai berikut:

- **Implementasi program** dalam bentuk **website**
- Sistem ***backend* wajib** menggunakan REST **API**
- Sanitasi input **sebaiknya** dilakukan di ***backend***
- ***Frontend*** disarankan menggunakan ***framework***
- Membuat **README** dan **dokumen** yang menjelaskan **cara kerja program**
- Mengerjakan **bonus** berupa **penerapan algoritma** lainnya

Kata-kata yang bercetak tebal adalah kata-kata yang dikenali oleh program dan mempunyai bobot tertentu berdasarkan implementasi (lebih lanjut dibahas di *Bab IV: Implementasi*). Pencocokan dilakukan dengan membagi tiap kata yang ada dalam masukan dan melakukan pencocokan tiap kata yang terdaftar dalam program dengan kata dalam masukan menggunakan algoritma pencocokan *brute force*.

Hasil pembobotan tiap masukan akan dimasukkan ke dalam sebuah *multidimensional array* sebagai persoalan utama dan diurutkan berdasarkan pembobotan yang diberikan. Pengurutan akan dilakukan menggunakan algoritma *merge sort* dengan membagi *multidimensional array* menjadi *array* yang lebih kecil hingga didapatkan upapersoalan berupa *array* dengan ukuran dua elemen. Setelah itu, akan dilakukan pengurutan berdasarkan pembobotan sehingga terdapat urutan pengerjaan yang harus dilakukan berdasarkan bobot yang dimiliki.

Sebuah pekerjaan juga dapat dikategorikan berdasarkan tingkat kesulitannya. Pengkategorian akan dilakukan ke tiga kelompok:

- *Important*: pekerjaan sebaiknya didahulukan dan mempunyai prioritas tertinggi. Misalnya, implementasi *frontend/backend*, visualisasi, *interface*, dan lain-lain
- *Moderate*: pekerjaan mempunyai prioritas sedang dan sebaiknya dikerjakan setelah pekerjaan dengan prioritas *important* telah dilakukan. Misalnya, pengujian implementasi, referensi ke dokumentasi, *bugtesting*, dan lain-lain
- *Backlog*: pekerjaan mempunyai prioritas rendah dan dapat dikerjakan menjelang waktu pengumpulan tugas. Misalnya, pembuatan video, README, atau bonus

Pembagian pekerjaan dilakukan dengan melihat rentang nilai maksimum dan minimum dari bobot yang ada. Rentang akan dibagi ke tiga buah bagian.

- Pekerjaan mempunyai prioritas *important* apabila nilai bobot berada di 1/3 nilai tertinggi dalam rentang
- Pekerjaan mempunyai prioritas *moderate* apabila nilai bobot berada di antara 1/3 nilai terendah dan 1/3 nilai tertinggi dalam rentang
- Pekerjaan mempunyai prioritas *backlog* apabila nilai bobot berada di 1/3 nilai terendah dalam rentang.

Pengkategorian ini dapat membantu pembagian prioritas pengerjaan tugas. Seorang anggota misalnya, dapat mengerjakan dua buah kebutuhan prioritas *important*, tiga buah kebutuhan prioritas *moderate*, dan sebuah kebutuhan prioritas *backlog*. Selain itu, pengkategorian ini mampu membantu menentukan prioritas mana yang sebaiknya didahulukan dan yang mana yang dapat dikerjakan nanti.

IV. IMPLEMENTASI

Implementasi dilakukan menggunakan bahasa pemrograman *Python* (versi 3.9.6). Implementasi dilakukan dengan membuat algoritma *merge sort* dan algoritma pencocokan *string*.

A. Algoritma Pencocokan String dan Penghitungan Bobot

Program menyimpan daftar kata-kata beserta bobotnya dalam struktur data *dictionary*, di mana *key* adalah nilai bobot yang ditambahkan dalam *query*. Nilai bobot merentang dari 1 hingga 5, dan semakin tinggi nilai bobot, semakin diprioritaskan kata tersebut. Nilai *value* dari tiap *key* adalah larik berisi kata-kata kunci dengan *value* tersebut.

Karena sifat pencocokan yang per-kata, algoritma *brute force* digunakan karena relatif lebih sederhana dan cocok untuk pencocokan sederhana. Pencocokan dilakukan dengan mencocokkan satu per satu huruf. Apabila tidak ditemukan kecocokan, maka pencocokan akan digeser satu-per-satu. Algoritma menandakan kata ditemukan apabila *iterator* pencocokan bernilai sama dengan panjang kata yang ingin dicocokkan.

```
keywords_dict = {
    1: ['laporan', 'dokumen', 'README', 'bonus', 'berkas',
        'demo', 'video', 'penjelasan', 'anggota', 'pembagian'],
    2: ['komentar', 'kode', 'code',
        'executable', 'modular', 'repository', 'folder', 'debugging',
        'bugtesting', 'instalasi', 'pemasangan', 'tahapan', 'pengaturan', 'belajar',
        'mempelajari'],
    3: ['algoritma', 'konsep', 'implementasi', 'penerapan', 'solusi', 'pengujian',
        'cara', 'kesesuaian', 'sesuai', 'materi', 'dataset', 'data', 'kreativitas', 'tutorial',
        'referensi', 'dokumentasi', 'pemakaian', 'penggunaan',
        'pengembangan', 'akses', 'desain'],
    4: ['bahasa', 'pustaka', 'library', 'CLI', 'GUI', 'web', 'desktop'
        'waktu', 'eksekusi', 'program', 'opsi', 'deploy', 'fitur',
        'buat', 'hasil', 'jenis', 'memakai', 'menggunakan', 'groundwork'],
    5: ['frontend', 'backend', 'framework', 'basis data', 'database', 'analisis',
        'aplikasi', 'pengujian', 'menguji', 'visualisasi', 'eksplorasi', 'perbaikan', 'wajib', 'benar',
        'interface'],
}
```

Gambar 5. *Dictionary* penyimpanan kata kunci

Source: Dokumen Pribadi

Program akan mengolah masukan dengan langkah berikut:

1. Membagi masukan kalimat ke larik berisi kata-kata menggunakan *.split(" ")* dan sifat *list comprehension* dari Python.

2. Menghitung bobot yang ada di tiap kata dengan mencocokan kata dari larik sebelumnya dengan kata-kata yang didaftarkan dalam *dictionary*.

3. Menggunakan algoritma *brute force* untuk melakukan pencocokan kata. Apabila kata ditemukan, bobot kata tersebut akan dicatat sementara. Apabila ditemukan kata yang mirip dengan bobot yang lebih besar di iterasi selanjutnya (misalnya, "dokumen" dan "dokumentasi" mempunyai bobot berbeda), maka bobot sementara akan diubah.

4. Melakukan iterasi berulang sampai semua kata dalam masukan telah diperiksa, dan mendaftarkan bobot total ke larik multidimensional.

5. Apabila masukan mengandung kata "bonus", kebutuhan dihitung sebagai kebutuhan bonus, sehingga total beban akan dibagi dua

Algoritma ini akan dilakukan terus-menerus hingga penulis tidak memasukkan masukan lagi ke dalam program. Setelah itu, program akan memasukkan larik multidimensional ke prosedur pengurutan.

Secara tidak langsung, proses penghitungan bobot ini memenuhi ikhtisar algoritma *divide and conquer*, yang melakukan pemecahan persoalan menjadi upapersoalan yang diinginkan dan menggabungkan solusi akhir. Sebuah masukan dipecah menjadi kata-kata tertentu dan pencocokan *string* dilakukan berdasarkan hasil pemecahan kata-kata tersebut. Dengan demikian, proses pencocokan dapat berlangsung dengan lebih efisien karena tidak perlu melakukan pencocokan terhadap masukan secara langsung. Hasil penghitungan bobot di setiap kata kemudian digabungkan untuk menjadi bobot total sebuah masukan. Meskipun tidak mengikuti proses pembagian secara rekursif, algoritma ini tetap mengikuti ikhtisar algoritma *divide and conquer*, yang melakukan penyelesaian masalah dengan melakukan penyelesaian terhadap upamasalah-upamasalah yang ada.

```python
def bfMatch(text, pattern):
    for i in range(len(text) - len(pattern) + 1):
        j = 0
        # melakukan iterasi selama j < len(pattern) dan kata-katanya cocok
        while (j < len(pattern) and pattern[j] == text[i + j]):
            j += 1

        # return True apabila iterasi berhasil dilakukan sampai habis
        if (j == len(pattern)):
            return True

    # default, return False apabila tidak ditemukan kecocokan
    return False
```

Gambar 6. Prosedur *string matching* dengan algoritma *brute-force*

Source: Dokumen Pribadi

```python

def calculateWeight(query):
    weight = 0

    # membagi query menjadi kata-kata
    split_query = [word.lower() for word in query.split(" ")]

    for word in split_query:
        temp = 0
        for i in range(len(keywords_dict.keys())):
            for j in range(len(keywords_dict[i + 1])):
                # melakukan pencocokan kata-kata dengan keyword tertentu
                if (len(word) >= len(keywords_dict[i + 1][j]) and bfMatch(word, keywords_dict[i + 1][j])):
                    if (i + 1 > temp):
                        # mengubah nilai weight sementara, apabila
                        # ditemukan kata yang lebih "cocok"
                        temp = i + 1

        # menambahkan bobot total
        weight += temp

    # apabila kebutuhan termasuk bonus,
    # kembalikan weight // 2 (bonus diprioritaskan terakhir)
    if "bonus" in split_query:
        return weight // 2

    # default, return weight biasa
    return weight

```

Gambar 7. Prosedur *penghitungan bobot*

Source: Dokumen Pribadi

B. Algoritma Pengurutan Pengerjaan

Algoritma pengurutan pengerjaan dilakukan dengan membuat prosedur *merge sort.* Prosedur *merge sort* menerima masukan berupa larik multidimensional berisi masukan-masukan kebutuhan dari pengguna beserta bobot yang dihitung menggunakan prosedur sebelumnya. Prosedur akan mengembalikan solusi berupa larik yang sudah terurut berdasarkan nilai beban yang dimiliki.

Langkah-langkah yang dijalankan pada prosedur *merge sort* adalah sebagai berikut:

1. Langkah *divide:* Apabila panjang larik lebih dari 1, bagi larik menjadi dua bagian berdasarkan titik tengah larik. Kedua larik akan disebut sebagai larik “kiri” dan larik “kanan”.
2. Langkah *conquer:* Panggil prosedur *merge sort* dengan parameter larik “kiri” dan larik “kanan”. Prosedur *merge sort* akan dieksekusi secara rekursif untuk larik “kiri” dan larik “kanan”. Prosedur akan diulangi terus-menerus hingga didapatkan upa-persoalan berupa larik berisi satu elemen saja.
3. Langkah *combine:* Lakukan prosedur penggabungan larik yang dipisah sehingga menghasilkan larik terurut dengan langkah-langkah berikut:
    a. Inisialisasi variabel *integer* i, j, dan k untuk melakukan iterasi terhadap larik dasar, larik “kiri” dan larik “kanan”
    b. Lakukan prosedur iterasi untuk membandingkan nilai bobot pada larik “kiri” dan larik “kanan”. Elemen dengan nilai bobot lebih tinggi akan dimasukkan ke larik terlebih dahulu. Iterasi akan dilakukan hingga salah satu antara larik “kiri” dan larik “kanan” telah diperiksa secara keseluruhan
    c. Ada kemungkinan apabila larik “kiri” dan larik “kanan” mempunyai perbedaan ukuran. Apabila ada elemen pada larik “kiri” atau “kanan” yang belum diperiksa, elemen tersebut langsung dimasukkan ke larik utama saja dengan asumsi bahwa nilai bobot pada elemen-elemen tersisa lebih kecil dibandingkan yang ada di larik sebelumnya.

```python
def merge_sort(array):
    if len(array) > 1:
        midpoint = len(array) // 2

        # melakukan pembagian array menjadi dua bagian
        # tergantung dengan nilai midpoint
        left = array[:midpoint]
        right = array[midpoint:]

        # memanggil prosedur merge_sort secara
        # rekursif (bagian divide)
        merge_sort(left)
        merge_sort(right)

        # menggabungkan dua bagian array
        i = 0
        j = 0
        k = 0

        # membandingkan nilai elemen kedua
        # dalam sebuah elemen, lalu mengurutkan
        # dari nilai terbesar ke terkecil
        while (i < len(left) and j < len(right)):
            if (left[i][1] > right[j][1]):
                array[k] = left[i]
                i += 1
            else:
                array[k] = right[j]
                j += 1
            k += 1

        # menggabungkan sisa array left
        while (i < len(left)):
            array[k] = left[i]
            i += 1
            k += 1

        # menggabungkan sisa array right
        while (j < len(right)):
            array[k] = right[j]
            j += 1
            k += 1

```

Gambar 8. Prosedur pengurutan secara *merge sort*

Source: Dokumen Pribadi

C. Program Utama

Prosedur program utama yang dibuat adalah sebagai berikut:

1. Menerima masukan pengguna berupa kebutuhan-kebutuhan yang ingin dikalkulasi. Setelah memasukkan kebutuhan, pengguna mengetikkan “exit” untuk menyudahi masukan dan melanjutkan prosedur
2. Memanggil prosedur *merge sort* untuk mengurutkan masukan. Setelah itu, prosedur akan menghitung batasan-batasan untuk membagi prioritas pengerjaan
3. Memberikan keluaran berupa daftar pekerjaan yang sudah diurutkan beserta nilai prioritas yang dimiliki per pekerjaan.

D. Pengujian

Pengujian akan dilakukan dengan masukan berikut yang terinspirasi dari Spesifikasi Tugas Besar 3 Mata Kuliah IF2211 Strategi Algoritma Tahun Ajaran 2021/2022:

- Implementasi *backend* menggunakan bahasa Golang
- Penerapan algoritma KMP sebagai opsi
- Penerapan algoritma Boyer-Moore yang sesuai dengan materi kuliah
- Pengaturan *backend* agar modular
- Pembuatan laporan beserta penjelasan dan pengujian
- Pembuatan bonus berupa video demo
- Berkas README yang mengandung pembagian pekerjaan dan cara eksekusi program
- Pengujian masukan menggunakan berkas plainteks
- Eksplorasi dokumentasi bahasa Golang
- Implementasi *database* menggunakan MySQL
- Instalasi pustaka *Echo* untuk *backend*
- Pengaturan akses basis data di *backend*
- Konsep algoritma *string matching*
- Implementasi *frontend* menggunakan
- Pembuatan bonus berupa *deployment*
- Kesesuaian algoritma yang dibuat dengan pengujian
- *Debugging* sebelum *deployment*
- Penerapan bonus berupa persentase kecocokan

```
Masukkan query: Implementasi backend menggunakan bahasa Golang
Masukkan query: Penerapan algoritma KMP sebagai opsi
Masukkan query: Penerapan algoritma Boyer-Moore yang sesuai dengan materi kuliah
Masukkan query: Pengaturan backend yang modular
Masukkan query: Pembuatan laporan beserta penjelasan dan pengujian
Masukkan query: Pembuatan bonus berupa video demo
Masukkan query: Berkas README yang mengandung pembagian pekerjaan dan cara eksekusi program
Masukkan query: Pengujian masukan menggunakan berkas plainteks
Masukkan query: Eksplorasi dokumentasi bahasa Golang
Masukkan query: Implementasi database menggunakan MySQL
Masukkan query: Instalasi pustaka Echo untuk backend
Masukkan query: Pengaturan akses basis data di backend
Masukkan query: Kesesuaian konsep algoritma string matching
Masukkan query: Implementasi frontend menggunakan framework React/Vue
Masukkan query: Pembuatan bonus berupa deployment
Masukkan query: Kesesuaian algoritma yang dibuat dengan pengujian
Masukkan query: Debugging sebelum deployment
Masukkan query: Penerapan bonus berupa persentase kecocokan
Masukkan query: exit
```

Gambar 9. Masukan pengujian 1

Source: Dokumen Pribadi

```
Urutan pengerjaan yang sebaiknya dilakukan:

IMPORTANT - Implementasi frontend menggunakan framework React/Vue - 17
IMPORTANT - Implementasi backend menggunakan bahasa Golang - 16
IMPORTANT - Kesesuaian algoritma yang dibuat dengan pengujian - 15
IMPORTANT - Pengaturan akses basis data di backend - 13
IMPORTANT - Berkas README yang mengandung pembagian pekerjaan dan cara eksekusi program - 13
IMPORTANT - Implementasi database menggunakan MySQL - 12
IMPORTANT - Eksplorasi dokumentasi bahasa Golang - 12
IMPORTANT - Penerapan algoritma Boyer-Moore yang sesuai dengan materi kuliah - 12
MODERATE - Instalasi pustaka Echo untuk backend - 11
MODERATE - Pembuatan laporan beserta penjelasan dan pengujian - 11
MODERATE - Pengujian masukan menggunakan berkas plainteks - 10
MODERATE - Penerapan algoritma KMP sebagai opsi - 10
MODERATE - Kesesuaian konsep algoritma string matching - 9
MODERATE - Pengaturan backend yang modular - 9
BACKLOG - Debugging sebelum deployment - 6
BACKLOG - Pembuatan bonus berupa deployment - 4
BACKLOG - Pembuatan bonus berupa video demo - 3
BACKLOG - Penerapan bonus berupa persentase kecocokan - 2
```

Gambar 10. Hasil keluaran pengujian 1

Source: Dokumen Pribadi

Dari hasil pengujian, tampak ada pengurutan daftar kebutuhan yang teracak menjadi daftar pekerjaan yang terurut sesuai bobot dan dapat dipertimbangkan sebagai alternatif solusi sebagai urutan pengerjaan sebuah tugas besar.

Pengujian lainnya akan dilakukan dengan masukan berikut yang terinspirasi dari Spesifikasi Tugas Besar 2 Mata Kuliah IF2211 Strategi Algoritma di tahun ajaran yang sama:

- Eksplorasi bahasa dan dokumentasi C# dan mempelajari Visual Studio
- Pembuatan desain frontend dalam ranah aplikasi desktop
- Pengembangan aplikasi dalam bentuk desktop
- Pengujian eksekusi di folder local
- Implementasi algoritma BFS dan DFS menggunakan bahasa C#
- Penerapan opsi pemilihan algoritma di frontend dan backend
- Hasil waktu eksekusi sebagai fitur GUI
- Instalasi NET framework
- Pengujian interface dan visualisasi yang benar
- Bugtesting dan pencocokan kesesuaian algoritma sebelum deployment
- Ada kesesuaian hasil pengujian dengan algoritma
- Pembuatan bonus berupa animasi visualisasi pohon solusi
- Pembuatan README yang menjelaskan cara kerja algoritma
- Eksplorasi algoritma yang sesuai materi kuliah

```
Masukkan query: Eksplorasi bahasa dan dokumentasi C# dan mempelajari Visual Studio
Masukkan query: Pembuatan desain frontend dalam ranah aplikasi desktop
Masukkan query: Pengembangan aplikasi dalam bentuk desktop
Masukkan query: Pengujian eksekusi di folder local
Masukkan query: Implementasi algoritma BFS dan DFS menggunakan bahasa C#
Masukkan query: Penerapan opsi pemilihan algoritma di frontend dan backend
Masukkan query: Hasil waktu eksekusi sebagai fitur GUI
Masukkan query: Instalasi NET framework
Masukkan query: Pengujian interface dan visualisasi yang benar
Masukkan query: Bugtesting dan pencocokan kesesuaian algoritma sebelum deployment
Masukkan query: Ada kesesuaian hasil pengujian dengan algoritma
Masukkan query: Pembuatan bonus berupa animasi visualisasi pohon solusi
Masukkan query: Pembuatan README yang menjelaskan cara kerja algoritma
Masukkan query: Eksplorasi algoritma yang sesuai materi kuliah
Masukkan query: exit
```

Gambar 11. Masukan pengujian 2

Source: Dokumen Pribadi

```
Urutan pengerjaan yang sebaiknya dilakukan:

IMPORTANT - Pengujian interface dan visualisasi yang benar - 20
IMPORTANT - Penerapan opsi pemilihan algoritma di frontend dan backend - 20
IMPORTANT - Pembuatan desain frontend dalam ranah aplikasi desktop - 17
IMPORTANT - Ada kesesuaian hasil pengujian dengan algoritma - 15
MODERATE - Eksplorasi algoritma yang sesuai materi kuliah - 14
MODERATE - Implementasi algoritma BFS dan DFS menggunakan bahasa C# - 14
MODERATE - Eksplorasi bahasa dan dokumentasi C# dan mempelajari Visual Studio - 14
MODERATE - Bugtesting dan pencocokan kesesuaian algoritma sebelum deployment - 12
MODERATE - Hasil waktu eksekusi sebagai fitur GUI - 12
MODERATE - Pengujian eksekusi di folder local - 11
BACKLOG - Pembuatan README yang menjelaskan cara kerja algoritma - 10
BACKLOG - Pengembangan aplikasi dalam bentuk desktop - 8
BACKLOG - Instalasi NET framework - 7
BACKLOG - Pembuatan bonus berupa animasi visualisasi pohon solusi - 6
```

Gambar 12. Hasil keluaran pengujian 2

Source: Dokumen Pribadi

Hasil pengujian menampilkan daftar pekerjaan yang sudah terurut. Program juga menampilkan bobot penghitungan yang dilakukan terhadap masukan berupa angka hasil penjumlahan

pembobotan. Selain itu, terdapat informasi tentang urgensi dari pekerjaan yang dilakukan:

- Pekerjaan yang mendesak (*important*), misalnya implementasi bagian *frontend* menggunakan *framework* React/Vue dan pengaturan akses basis data di *backend*, mempunyai bobot pengerjaan antara 12 dan 17
- Pekerjaan yang penting namun dapat dikerjakan setelah pekerjaan mendesak (*moderate*), misalnya instalasi pustaka *Echo* untuk *backend* dan pengaturan *backend* yang modular. mempunyai bobot pengerjaan antara 7 hingga 11
- Pekerjaan yang dapat dilakukan setelah pekerjaan lainnya selesai (*backlog)*, misalnya *debugging* sebelum proses *deployment* dan pembuatan bonus-bonus, mempunyai bobot pengerjaan antara 2 hingga 6

Dengan informasi demikian, seorang mahasiswa dapat menentukan urutan pekerjaan dalam sebuah tugas besar yang dilakukan. Sebuah kelompok juga dapat melakukan pembagian tugas menggunakan informasi urgensi yang harus dilakukan secara merata sehingga setiap orang dapat berpartisipasi dengan aktif dalam pengerjaan tugas besar tersebut. Misalnya, setiap mahasiswa dalam sebuah kelompok bertanggung jawab atas pengerjaan dua kebutuhan *important*, dua kebutuhan *moderate*, dan dua kebutuhan *backlog*.

Perlu ditekankan bahwa sifat implementasi yang dibuat adalah sebuah sugesti yang didasarkan pada perhitungan bobot secara objektif, sehingga tidak selalu harus diikuti oleh pengguna. Implementasi yang dibuat juga dicocokkan dengan program yang dibuat sebagai pembuktian dari konsep algoritma yang dibuat untuk menyelesaikan masalah tersebut. Pengembangan lebih lanjut dapat menerapkan *natural language processing* untuk menentukan bobot berdasarkan kata-kata yang diterima secara alamiah tanpa menggunakan pencocokan kata-kata secara tertulis dengan *dictionary*.

V. Kesimpulan dan Saran

A. Kesimpulan

Tugas besar dapat menjadi sebuah tantangan bagi seorang mahasiswa apabila seorang mahasiswa tidak dapat menentukan prioritas pembagian kebutuhan-kebutuhan yang dilampirkan dalam sebuah spesifikasi. Dengan memanfaatkan algoritma *divide and conquer*, kita dapat mengurutkan sebuah daftar kebutuhan berdasarkan bobot yang diasosiasikan terhadap masukan tersebut. Pembobotan sebuah masukan kebutuhan dihitung menggunakan algoritma pencocokan *string* dan akumulasi hasil perhitungan yang mempunyai ikhtisar *divide and conquer*.

Langkah-langkah menentukan urutan pengerjaan yang dilakukan adalah sebagai berikut:

1. Menghitung bobot tiap kebutuhan yang ada menggunakan algoritma pencocokan *string* dan membandingkan kata-kata yang ada di dalam

2. Mengurutkan masukan-masukan yang dimasukkan berdasarkan bobot yang ada menggunakan algoritma *merge sort*

3. Mengklasifikasikan tipe prioritas dari masukan sehingga dapat diketahui tingkat urgensi dari sebuah kebutuhan

Dengan melakukan langkah-langkah tersebut, kita dapat mengerjakan suatu tugas besar dengan lebih terarah dengan adanya pengetahuan mengenai kebutuhan-kebutuhan tugas besar yang lebih terstruktur. Selain itu, pembagian tanggung jawab dari kebutuhan-kebutuhan juga dapat dilakukan berdasarkan tingkat urgensi dari sebuah kebutuhan, sehingga pengerjaan dapat dilakukan dengan lebih merata.

B. Saran

Penulis menyarankan pembobotan sebuah masukan sebaiknya dilakukan menggunakan prinsip *natural language processing*, yang memungkinkan program untuk mengenali kata-kata yang ada dan melakukan pembobotan sendiri sesuai kata tersebut tanpa harus dituliskan secara fisik ke sebuah *dictionary*. Karena titik pusat pengerjaan makalah ini adalah dalam pemanfaatan algoritma, maka penulis membuat implementasi pembobotan menggunakan kata-kata yang dibobotkan secara manual dalam sebuah *dictionary*.

Pranala Video

Video yang dibuat dapat disaksikan di pranala berikut: https://youtu.be/YLviiHAkAxA

Sampel Kode

Sampel kode dapat diakses di pranala berikut: https://github.com/clumsyyyy/Tubes-Weighter

Ucapan Terima Kasih

Puji syukur penulis ucapkan kepada Tuhan Yang Maha Esa. Atas berkat dan rahmat-Nya, maka makalah "Aplikasi Algoritma Divide and Conquer dan Pencocokan String dalam Pembagian Prioritas Kebutuhan dalam Spesifikasi Tugas Besar" ini dapat diselesaikan dengan tepat waktu.

Penulis juga mengucapkan terima kasih kepada Ibu Dr. Masayu Leylia Khodra, S.T, M.T.sebagai dosen pengajar Mata Kuliah IF2211 Strategi Algoritma Kelas 01 Tahun Ajaran 2021/2022 atas bimbingan dan pengajaran yang tak henti-hentinya dilakukan di kelas terkait teori-teori dasar yang ada di makalah ini. Penulis juga mengucapkan terima kasih kepada Bapak Dr. Ir. Rinaldi Munir, M.T. Beliau telah mengelola situs berisi referensi dan sumber pembelajaran dan spesifikasi-spesifikasi tugas besar untuk mata kuliah IF211 Strategi Algoritma bagi penulis dan sesama mahasiswa Teknik Informatika ITB lainnya.

Terakhir, penulis ingin mengucapkan terima kasih kepada orang tua, keluarga, dan rekan-rekan sesama mahasiswa Teknik Informatika yang telah mendukung dan menyemangati penulis dalam pengerjaan makalah ini.

DAFTAR PUSTAKA

1. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf. Diakses 4 Mei 2022

2. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian2.pdf. Diakses 4 Mei 2022

3. Tugas Besar 3 IF2211 Strategi Algoritma 2022.docx (itb.ac.id). Diakses 6 Mei 2022

4. Tugas Besar II IF2211 Strategi Algoritma Semester II Tahun 2020/2021: Pengaplikasian Algoritma BFS dan DFS dalam Implementasi Folder Crawling.

5.. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf. Diakses 19 Mei 2022

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, 19 Mei 2022

Owen Christian Wijaya - 13520124