Aplikasi *Knuth-Morris-Pratt Algorithm* pada Pencarian Isi Dokumen *Lokal*

Seperayo - 13516068
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13516068@std.stei.itb.ac.id

Abstrak*—Makalah ini membahas suatu ide tentang implementasi dari algoritma KMP sebagai sebuah algoritma pencocokan string dalam suatu pencarian dokumen berdasarkan kata kunci yang ada di dalam dokumen itu sendiri. Pencocokan dilakukan dengan melakukan ekstrak string dari setiap dokumen dalam suatu folder. Hasil ekstrak ini kemudian disamakan dengan masukan kata kunci dari pengguna. Kemudian akan dikeluarkan daftar dokumen yang mengandung kata kunci masukan pengguna di dalamnya. Melalui metode ini, akan didapatkan suatu list dokumen yang di dalamnya terdapat sejumlah kata kunci yang dicari dari sejumlah dokumen lokal yang ada. Implementasi dilakukan dengan menggunakan bahasa pemrograman Java dan kakas Netbeans 8.2, selain itu juga akan digunakan library tambahan berupa Apache POI untuk mempermudah proses ekstraksi. Fokus pada program sampel adalah pencarian dengan algoritma KMP dan format file docx.

***Kata Kunci*—Knuth-Morris-Pratt, Dokumen, Lokal, String, List.**

I. PENDAHULUAN

Dokumen adalah suatu bagian dari kehidupan kita sehari-hari, mulai dari kalangan pelajar hingga para pekerja kantoran atau bahkan toko-toko kecil di pasar. Dokumen digunakan dalam melakukan banyak hal, seperti melakukan pencatatan suatu materi, sebagai media untuk melaporkan hasil suatu kegiatan (eksperimen, perbankan, tugas, dst), perantara komunikasi antar dua pihak yang melakukan transaksi, dst.

Pada masa ini banyak di antara kita membuat dokumen dengan menggunakan bantuan perangkat lunak atau elektronik. Penggunaan teknologi dalam pembuatan suatu dokumen meningkatkan efisiensi dan keefektifan dari proses dokumentasi. Hal ini belum lagi ditambah dengan adanya alat cetak berupa *printer* yang sangat meningkatkan produktivitas kita saat ini.

Perihal navigasi antar dokumen menjadi sangat penting kala kita memiliki dokumen dalam jumlah yang banyak. Kita perlu untuk menyusun dokumen itu demi mempermudah pencarian dokumen referensi ataupun dokumen penting yang sebelumnya telah kita simpan. Proses pencarian yang kita lakukan biasanya adalah menggunakan nama dokumen sebagai media yang dicari.

Hal ini dapat menyelesaikan sebagian masalah dalam proses navigasi dokumen. Namun tak jarang, bagian yang kita ingin dapatkan dari dokumen yang kita cari bukan terletak pada nama dokumen. Akan tetapi pada isi dari dokumen itu sendiri, proses pencarian pun akan menjadi sangat tidak efektif. Kita harus melakukan pengecekan dengan membuka satu demi satu dokumen yang berkemungkinan sebagai yang kita cari.

Melalui latar belakang itu, makalah ini akan membahas ide untuk melakukan ekstraksi teks dari dalam setiap dokumen dalam menemukan dokumen yang kita cari. Hasil dari ekstrak ini akan berbentuk list dengan dokumen-dokumen yang mengandung kata kunci pencarian. Proses ini menggunakan pendekatan pencocokan string dengan algoritma KMP atau Knuth-Morris-Pratt sebagai media yang mencocokan.

Algoritma ini dipilih karena kemudahannya dalam proses implementasi dan akurasi yang cukup memuaskan dalam penggunaan tingkat sederhana. Proses pencarian juga efektif dan efisien dalam menangani string yang mengandung banyak variasi alfabet. Hasil pencarian akan memuat *path* dari setiap dokumen, sehingga memudahkan kita nantinya untuk membuka dokumen itu dengan lokasi yang telah disediakan.

Sedangkan untuk ekstraksi dokumen digunakan *library* tambahan pada Java, yaitu Apache POI. Komponen tambahan ini merupakan API yang banyak digunakan dalam melakukan modifikasi terhadap dokumen dari *Microsoft Office* dalam bahasa Java. Program sampel yang dibuat dalam makalah ini akan fokus pada ekstraksi dan menggunakan file dengan format docx sebagai media pencarian.

Penulisan makalah ini bertujuan untuk memberikan ide pencarian yang berbeda dari pencarian dokumen pada biasanya. Penggunaan informasi ini diharapkan dapat bermanfaat bagi pembaca dalam memanfaatkan pencarian dokumen dengan menggunakan isi dokumen sebagai media yang dicari.

II. LANDASAN TEORI

Dalam melakukan pemecahan masalah ini diperlukan beberapa pengetahuan dasar, yaitu:

2.1. Algoritma Pencocokan String

Dalam melakukan proses pencarian suatu string dengan menggunakan sebuah kata kunci masukan, terdapat beberapa algoritma pencarian yang dapat digunakan, seperti:

1. *Brute Force*

Algoritma pencocokan string dengan menggunakan metode *brute force* ini dapat memberikan hasil paling akurat, namun layaknya algoritma *brute force* lainnya memiliki kekurangan pada performa waktu eksekusinya. Penggunaan metode ini dapat dilakukan dengan melakukan penggeseran sebanyak 1 indeks untuk setiap kali array string media pencarian dan kata kunci memiliki ketidakcocokan.[4]

Dalam tingkat pencarian yang sederhana dimana tidak mengikutsertakan media pencarian dengan jumlah yang tinggi,

metode *brute force* tidak akan memiliki perbedaan yang jauh dengan algoritma lainnya. Namun, jika media pencarian berjumlah banyak tentu pergeseran 1 indeks untuk setiap ketidakcocokan yang ditemukan ini akan memberikan dampak negatif pada performa pencarian yang dilakukan.

Algoritma ini memiliki kompleksitas sebesar O(mn) untuk kasus terburuk pencarian, O(n) untuk kasus terbaik pencarian, dan O(m+n) dalam kasus rata-rata pencarian.

Gambar 2.1.1 Proses Pencarian dengan *Brute Force*[1]

Pada gambar 2.1.1 dapat dilihat proses pencarian dengan metode *brute force* melakukan pergeseran 1 indeks untuk setiap ketidakcocokan antara media pencarian dan kata kunci.

2. *Knuth Morris Pratt*

Algoritma pencocokan string ini telah dikenal sejak tahun 1977 secara umum, dimana James H. Morris, Vaughan R. Pratt, dan Donald E. Knuth mempublikasikan hasil pekerjaannya ke publik. Algoritma KMP atau juga dikenal sebagai algoritma Knuth-Morris-Pratt merupakan sebuah algoritma pencarian yang memulai proses pencarian dari arah kiri ke kanan, sesuai dengan pendekatan pada metode *brute force*.[3]

Namun yang membedakan kedua algoritma ini adalah kemampuan KMP untuk melakukan lompatan dengan jumlah indeks lebih dari 1 dalam memanfaatkan pola dalam kata kunci itu sendiri. Lompatan indeks yang lebih tinggi ini memberikan dampak yang positif terhadap peningkatan performa pencarian dibandingkan dengan algoritma *brute force*.

Proses lompatan menggunakan jarak terjauh dari indeks prefiks dan sufiks kata kunci yang berulang. Oleh sebab itu, algoritma ini melakukan proses pencarian pola pada kata kunci pencarian terlebih dahulu, kemudian data penyelidikan banyak lompatan yang bisa dilakukan ini akan diterapkan saat melakukan pencocokan dengan media pencarian.[5]

Sebagai ilustrasi dari algoritma KMP, algoritma akan mencari ketidakcocokan pada kata kunci dengan posisi **j**, lalu dicatat pula posisi sebelum ketidakcocokan itu terjadi atau **k**. Lompatan atau **b(k)** didapatkan dari ukuran indeks antara prefiks terbesar pada kata kunci (jangkauan dari 0 hingga k) dengan sufiks (jangkauan dari 1 hingga k). Jika terdapat kata kunci berupa abababbabca, maka akan di dapat data berikut:

j	0	1	2	3	4	5	6	7	8	9
P[*j*]	a	b	a	b	a	b	a	b	c	a
k	-	0	1	2	3	4	5	6	7	8
b[*k*]	-	0	0	1	2	3	4	5	6	0

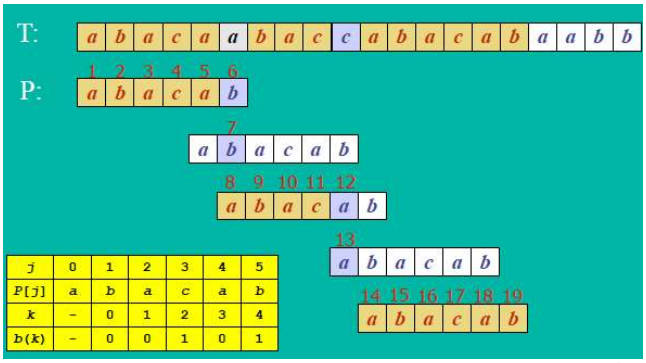
Gambar 2.1.2 Pendataan Pola pada Kata Kunci dengan KMP *Algorithm*[1]

Dapat dilihat dari gambar 2.1.2 bahwa **k** sebagai posisi sebelum terjadinya ketidakcocokan adalah sebesar **j-1**. Pada **b(4)** kita dapat memahami bahwa data itu merepresentasikan prefix terbesar dari 0 hingga 4 yang juga merupakan suffix kata kunci dari 1 hingga 4. Gambar diatas memiliki **b(4)** dengan prefix ababa dan sufiks baba, dimana terdapat pengulangan aba karenanya **b(4)** memiliki hasil indeks 2 di kata kunci.

Algoritma KMP sendiri memiliki keunggulan pada gerakan pencariannya yang terus maju tanpa memerlukan *backtrack* pada media pencarian. Alur pencarian ini memiliki dampak pada potensi KMP dalam melakukan pencarian pada media yang besar dalam hal ukurannya. Sedangkan kelemahan dari algoritma ini adalah ketika kata kunci yang dimasukkan memiliki jumlah yang banyak.

Hal ini terjadi karena semakin banyak huruf atau nomor pada kata kunci pencarian, maka semakin tinggi pula kemungkinan terjadinya ketidakcocokan pada awal kata kunci. KMP sendiri bekerja lebih baik saat ketidakcocokan terjadi pada akhir kata kunci.

Algoritma KMP memiliki kompleksitas waktu *border function* sebesar O(m) dan pencarian string sebesar O(n). Oleh karena itu, kompleksitas waktu dari algoritma KMP secara keseluruhan adalah O(m+n). Berikut adalah ilustrasi proses pergeseran yang dilakukan oleh algoritma KMP selama pencarian berlangsung:



Gambar 2.1.3. Proses Pergeseran pada KMP *Algorithm*[1]

3. *Boyer Moore*

Algoritma ini diperkenalkan oleh Bob Boyer dan J.S. Moore pada tahun 1977, berkebalikan dengan pendekatan pada algoritma KMP, algoritma *Boyer-Moore* ini melakukan pencarian dari arah kanan ke kiri pada kata kunci. Algoritma ini menggunakan 2 teknik secara umum, yaitu:

1. *The Character-Jump Technique*

Teknik ini melakukan aksi perbandingan antar 2 karakter yang berbeda satu sama lain. Jika terdapat suatu karakter **x** pada media pencarian dengan indeks **i** yang mengalami ketidakcocokan dengan kata kunci pencarian pada indeks **j**, maka akan dilakukan aksi sesuai kasus-kasus berikut:

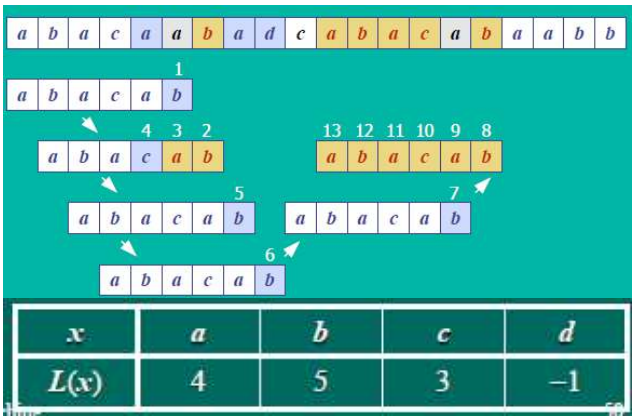
(1) Ketika karakter **x** di kata kunci dengan indeks **k** berada sebelum indeks **i** di media pencarian, maka pencarian digeser sebesar **j-k**. (Sehingga **k** dan **I** paralel)
(2) Ketika karakter **x** di kata kunci dengan indeks **k** berada setelah indeks **I** di media pencarian, maka kata kunci digeser sebesar 1 indeks.
(3) Ketika karakter **x** tidak dimiliki kata kunci, maka kata

kunci digeser sebesar **j+1**. Sehingga kata kunci di indeks 0 paralel dengan media pencarian di indeks **i+1**.

2. *The Looking-Glass Technique*

Teknik ini digunakan saat membandingkan karakter akhir pada kata kunci (indeks **j**) dengan karakter pada media pencarian (indeks **i**). Jika karakter sama, maka akan dilanjutkan ke karakter **j-1** dan **i-1**, dan seterusnya.[1]

Oleh sebab itu, algoritma ini akan melakukan pencatatan terhadap kemunculan terakhir suatu karakter di dalam kata kunci **L()**. **L()** akan bernilai -1 jika karakter itu tidak dimiliki oleh kata kunci pencarian. Berikut contoh penccocokan dengan menggunakan algoritma *Boyer Moore*:



Gambar 2.1.4 Proses Pergeseran pada *Boyer Moore Algorithm*[1]

Pada gambar 2.1.4 dapat dilihat proses pencocokan pertama kata kunci memiliki karakter b dan media pencarian a. Kemudian kemunculan terakhir dari a pada kata kunci adalah 1 indeks dibelakang b, oleh karena itu kata kunci digeser sebesar 1. Hal yang sama terjadi sebanyak 2 kali setelahnya, pada pencocokan keempat karakter d tidak ada di kata kunci. Pergeseran dilakukan sebesar 6 indeks (5+1).

Pencocokan terakhir karakter a di media pencarian dan b di kata kunci, sama halnya dengan pencocokan pertama digeser 1 indeks. Algoritma *Boyer Moore* ini memiliki kompleksitas waktu sebesar O(nm + A) untuk kasus terburuk pencarian. Namun dalam keunggulan dari algoritma ini adalah ketika kata kunci pencarian berjumlah besar.

2.2. Apache POI

Library tambahan ini digunakan dalam proses ekstraksi dari dokumen. Komponen ini merupakan komponen publik yang di distribusi oleh *Apache Software Foundation* dengan tujuan untuk melakukan desain dan modifikasi terhadap file *Microsoft Office* dalam bahasa pemrograman Java. Dalam komponen ini terdapat *method* dan kelas yang memberikan kemampuan untuk mengatur input data atau file dari *Microsoft Office*.[2]

Library ini dipilih karena tingkat penggunaan produk *Microsoft Office* yang tinggi dikalangan masyarakat. Apache POI sendiri memiliki komponen-komponen lain di dalamnya. Berikut adalah daftar komponen yang ada dalam API ini:

1. POIFS

   Singkatan dari *Poor Obfuscation Implementation File System*, komponen ini merupakan komponen dasar dari POI yang digunakan dalam membaca file yang berbeda secara eksplisit.
2. HSSF

   Horrible Spreadsheet Format merupakan komponen yang digunakan *editing* file *Microsoft Office Excel* yang formatnya xls.
3. XSSF

   XML Spreadsheet Format digunakan untuk melakukan modifikasi file *Excel* dengan format xlsx dari *Microsoft Office*.
4. HPSF

   Horrible Property Set Format digunakan dalam melakukan ekstraksi himpunan properti dari file *Microsoft Office*.
5. HWPF

   Horrible Word Processor Format digunakan untuk melakukan *editing* file *Microsoft Office Word* dengan format doc.
6. XWPF

   XML Word Processor Format digunakan untuk membaca dan menulis, serta memodifikasi file *Word* dengan format docx dari *Microsoft Office*.
7. HSLF

   Horrible Slide Layout Format digunakan untuk membaca, membuat, dan mengedit file presentasi PowerPoint dari *Microsoft Office*.
8. HDGF

   Horrible DiaGram Format merupakan komponen yang mengandung kelas dan *method* untuk file *Microsoft Office Visio binary*.
9. HPBF

   Horrible PuBlisher Format digunakan untuk membaca, menulis, dan melakukan modifikasi pada file *Microsoft Office Publisher*.

Dalam makalah ini contoh ekstraksi yang dilakukan menggunakan komponen XWPF atau dokumen dengan ekstensi docx. Berikut adalah kelas dari Apache POI yang digunakan dalam progam sampel demi membuktikan ide makalah:

(1) XWPFDocument

Kelas ini merupakan bagian dari org.apache.poi.xwpf.usermodel yang digunakan dalam memasukkan data dokumen di lokasi pencarian ke dalam program. *Method* yang dimilikinya adalah commit() untuk menyimpan dokumen, createParagraph() untuk malakukan *append* paragraf, createTable() untuk membuat tabel kosong, createTOC() untuk membuat daftar isi, getParagraphs() untuk mengembalikan paragraf dengan *header/footer*, getStyle() untuk mengembalikan *styles* objek, dst.

(2) XWPFWordExtractor

Kelas ini merupakan bagian dari org.apache.poi.xwpf.extractor dimana berfungsi sebagai *parser* dasar untuk mengekstrak teks sederhana dari *Microsoft Word*. Method kelas ini adalah getText() untuk mengembalikan seluruh teks dari dokumen.

III. IMPLEMENTASI DAN EKSPERIMEN

3.1. Penjelasan Singkat Alur Program Sampel

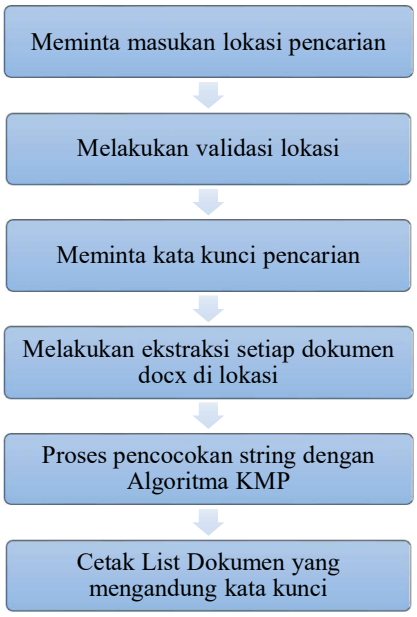
Program sampel memiliki 4 kelas, yaitu ReadDocx, FileLoader, KMPMatcher, dan StringMatcher. Kelas StringMatcher digunakan untuk method yang melakukan pencarian secara *case sensitive*. Kelas ReadDocx digunakan untuk proses ekstraksi dokumen dengan format docx. Anak dari kelas StringMatcher adalah KMPMatcher yang mengimplementasikan pencocokan string dengan menggunakan algoritma KMP.

Sedangkan kelas FileLoader digunakan sebagai main yang mengontrol alur program dan penggunaan kelas lainnya. Berikut adalah bagian import dari program yang akan digunakan dalam mengolah ekstraksi file dengan format docx:

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import org.apache.poi.xwpf.extractor.XWPFWordExtractor;
import org.apache.poi.xwpf.usermodel.XWPFDocument;
```

Gambar 3.1. Bagian Import dari Program Sampel

Dalam proses kerja program sampel akan menggunakan langkah-langkah berikut:



Gambar 3.1.2. Proses Kinerja Program Sampel

Proses diawali dengan melakukan permintaan masukan lokasi pencarian berupa *path*. Lokasi masukan akan diperiksa apakah mengandung file dengan format docx atau tidak, lalau apakah benar lokasi itu ada. Kemudian program sampel akan meminta kata kunci pencarian, selanjutnya program akan langsung memanfaatkan API Apache POI untuk mengekstrak dokumen-dokumen dengan format docx di lokasi masukan. Setelah ekstrak berhasil dengan menggunakan KMP *Algorithm* kata kunci masukan dicocokan dengan hasil ekstrak. Keluaran dari program sampel adalah daftar dari dokumen yang mengandung kata kunci pencarian dalam bentuk lokasi di komputer.

3.2. Pemanfaatan *Library* File dan POI

Dalam memasukkan data dokumen, program sampel menggunakan FileInputStream, XWPFDocument, dan XWPFWordExtractor. Berikut tampilan pada program sampel:

```
String temp = file.getName().toString();
XWPFDocument docx = new XWPFDocument(new FileInputStream(temp));

//Search pattern text
XWPFWordExtractor we = new XWPFWordExtractor(docx);
String text = we.getText();
```

Gambar 3.2.1. Pemanfaatan *Library POI*

Proses ekstraksi dilakukan untuk setiap dokumen dengan format docx, kemudian dokumen yang terbukti mengandung kata kunci akan dimasukkan ke dalam list yang nantinya dicetak di akhir eksekusi program.

List keluaran program sampel akan menuliskan “No such pattern found in the folder” ketika di lokasi tidak ada dokumen yang memuat kata kunci. File yang dimasukkan menggunakan getName().toString() untuk mendapatkan nama file yang akan diekstrak, sedangkan dalam penulisan lokasi dokumen nantinya digunakan getCanonicalPath().toString() seperti pada gambar berikut:

```
//if found add to toggler list
if (count>0){
    temp = file.getCanonicalPath().toString();
    toggler.add(temp);
```

Gambar 3.2.2. Penulisan Daftar Lokasi Dokumen

3.3. Uji Coba Program Sampel

Pada pengujian ini akan difokuskan pada tahap ekstraksi dimana variasi pencocokan kata kunci dalam dokumen yang menjadi perhatian utama. Berikut beberapa uji coba yang dilakukan dengan menerapkan ide dari makalah ini dan program sampel:

1. Uji Coba 1

Kata kunci yang digunakan pendek dengan isi dokumen sebanyak 150-200 kata dan jumlah dokumen sebanyak 10. Kata kunci yang digunakan adalah process, dengan dokumen yang mengandung kata ini adalah dokumen Uji 1-1, Uji 1-4, Uji 1-5, Uji 1-6, Uji 1-9, dan Uji 1-10.

```
Enter the path to folder to search for files
D:\Seperayo\Kuliah\Strategi_Algoritma\Makalah\ReadDocx

Enter pattern sample =
process

Ditemukan pada =
D:\Seperayo\Kuliah\Strategi_Algoritma\Makalah\ReadDocx\Uji 1-1.docx
D:\Seperayo\Kuliah\Strategi_Algoritma\Makalah\ReadDocx\Uji 1-10.docx
D:\Seperayo\Kuliah\Strategi_Algoritma\Makalah\ReadDocx\Uji 1-4.docx
D:\Seperayo\Kuliah\Strategi_Algoritma\Makalah\ReadDocx\Uji 1-5.docx
D:\Seperayo\Kuliah\Strategi_Algoritma\Makalah\ReadDocx\Uji 1-6.docx
D:\Seperayo\Kuliah\Strategi_Algoritma\Makalah\ReadDocx\Uji 1-9.docx
```

Gambar 3.3.1. Hasil Uji Coba 1

Hasil pengujian sama dengan hasil yang diharapkan, hal ini membuktikan bahwa program sampel berjalan dengan baik dan

ide dari makalah ini dapat diimplementasikan secara sempurna. Waktu eksekusi program adalah 0,63 detik.

2. Uji Coba 2

Kata kunci pada pengujian ini adalah sebuah kalimat berupa "The absorption of a photon by the antenna complex frees an electron". Kalimat ini dijumpai pada dokumen Uji 1-1 dan Uji 1-10.

```
Enter the path to folder to search for files
D:\Seperayo\Kuliah\Strategi_Algoritma\Makalah\ReadDocx

Enter pattern sample =
The absorption of a photon by the antenna complex frees an electron

Ditemukan pada =
D:\Seperayo\Kuliah\Strategi_Algoritma\Makalah\ReadDocx\Uji 1-1.docx
D:\Seperayo\Kuliah\Strategi_Algoritma\Makalah\ReadDocx\Uji 1-10.docx
```

Gambar 3.3.2. Hasil Uji Coba 2

Melalui hasil eksekusi dapat dilihat bahwa program memiliki kemampuan untuk mencari sebuah kalimat di media pencarian. Hasil eksekusi yang sama dengan hasil yang diharapkan menunjukkan ide pencarian isi dokumen berhasil dilaksanakan. Waktu eksekusi sebesar 1,84 detik.

3. Uji Coba 3

Kali ini pengujian dilakukan dengan menggunakan kata kunci yang tidak berada di dokumen pengujian. Media pencarian sama dengan 2 pengujian sebelumnya.

```
Enter the path to folder to search for files
D:\Seperayo\Kuliah\Strategi_Algoritma\Makalah\ReadDocx

Enter pattern sample =
xxxxxxx

No such pattern found in the folder
```

Gambar 3.3.3. Hasil Uji Coba 3

Hasil eksekusi menunjukkan bahwa program sampel yang menerapkan ide makalah ini berhasil mengatasi kasus ketika kata kunci tidak dikandung oleh dokumen di lokasi pencarian. Waktu eksekusi adalah 0,38 detik

4. Uji Coba 4

Pengujian keempat dilakukan dengan menggunakan lokasi yang tidak benar. Lokasi disk diubah menjadi Disk C.

```
Enter the path to folder to search for files
C:\Seperayo\Kuliah\Strategi_Algoritma\Makalah\ReadDocx

There is no Folder @ given path :C:\Seperayo\Kuliah\Strategi_Algoritma\Makalah\ReadDocx
```

Gambar 3.3.4. Hasil Uji Coba 4

Hasil eksekusi menunjukkan bahwa pencarian isi dokumen dengan menggunakan *path* yang tidak nyata dapat diatasi. Hal ini dibuktikan dengan adanya pesan kesalahan dari program sampel. Waktu eksekusi program sampel adalah 0,1 detik.

5. Uji Coba 5

Pengujian menggunakan kata kunci "error" dan dokumen pengujian memiliki panjang sebesar 3000 kata dengan jumlah 10 buah. Hasil yang diharapkan adalah dokumen Uji 2-1, Uji 2-3, dan Uji 2-5.

```
Enter the path to folder to search for files
D:\Seperayo\Kuliah\Strategi_Algoritma\Makalah\ReadDocx

Enter pattern sample =
error

Ditemukan pada =
D:\Seperayo\Kuliah\Strategi_Algoritma\Makalah\ReadDocx\Uji 2-1.docx
D:\Seperayo\Kuliah\Strategi_Algoritma\Makalah\ReadDocx\Uji 2-3.docx
D:\Seperayo\Kuliah\Strategi_Algoritma\Makalah\ReadDocx\Uji 2-5.docx
```

Gambar 3.3.5. Hasil Uji Coba 5

Hasil eksekusi dan hasil yang diharapkan sama dengan waktu eksekusi sebesar 1,12 detik. Program sampel berhasil mencari dokumen dangan media pencarian bervolume tinggi.

IV. ANALISIS

Melalui hasil 4 hasil uji yang dilakukan pada subbab sebelumnya, program sampel dapat dinyatakan berjalan sesuai dengan harapan. Pencarian dengan menggunakan sebuah kata kunci ataupun sebuah kalimat dapat menghasilkan hasil pencarian dokumen yang benar.

Waktu eksekusi juga menunjukkan sifat dari algoritma KMP, dimana ketika menggunakan kata kunci yang pendek pada uji coba 1 kita dapat menemukan hasil dalam waktu 0.43 detik, sedangkan saat kita menggunakan kata kunci yang panjang waktu yang diperlukan sebesar 1.84 detik.

Perbedaan sejauh 1,41 detik ini cukup besar, karena dokumen pengujian yang digunakan tidaklah panjang dalam isinya dan jumlah dokumen yang dicari hanya sebanyak 10 buah. Hal ini mencerminkan sifat algoritma KMP dimana penggunaan kata kunci yang pendek memberikan hasil pencarian dengan performa yang lebih tinggi dibandingkan dengan kata kunci yang panjang.

Perubahan performa ini diakibatkan oleh algoritma KMP menggunakan pengulangan pola pada kata kunci. Kata kunci yang panjang meningkatkan kemungkinan ditemukannya ketidakcocokan antara media pencarian dan kata kunci. Selain itu, pencarian dengan media pencarian berupa isi dokumen tidak akan memberikan hasil error.

Hal ini dapat terjadi dengan menggunakan Exception dan *handler* terhadap kasus-kasus tersebut. Sebagai contoh pada uji coba 3, kata kunci yang memang tidak dimiliki oleh setiap dokumen di lokasi pencarian akan memberikan pesan kesalahan. Begitu pula dengan memasukkan *path* lokasi pencarian yang tidak benar dapat diatasi dan memberikan pesan kesalahan.

Kedua uji coba itu (3 dan 4) menunjukkan bahwa pencarian kata kunci pada isi dokumen merupakan ide yang dapat direalisasikan tanpa adanya kemungkinan kesalahan hasil pencarian. Pengujian terakhir dilakukan untuk menguji kemampuan algoritma KMP dalam melakukan pencocokan dengan media pencarian bervolume besar.

Hasil dari uji coba 5 mencerminkan sifat algoritma KMP dimana algoritma ini memiliki kemampuan yang baik dalam mencocokan string ke media pencarian dengan data yang

banyak. Ide makalah juga menunjukkan hasil yang terbukti, dimana dokumen yang panjang juga masih dapat dilakukan pencarian isinya.

V. KESIMPULAN

Melalui pengerjaan makalah dengan judul "Aplikasi *Knuth-Morris-Pratt Algorithm* pada Pencarian Isi Dokumen Lokal" dapat disimpulkan bahwa ide makalah dapat diaplikasikan dan efektif dalam eksekusinya. Pencarian isi dokumen memberikan hasil yang diharapkan berdasarkan pengujian pada subbab III, dimana dokumen yang besar isinya ataupun kesalahan pada masukan lokasi atau kata kunci dapat diatasi dan tidak memberikan hasil yang salah.

Algoritma KMP dalam kerjanya di pencarian isi dokumen memberikan hasil yang memuaskan. Algoritma ini cocok untuk kasus masalah pada makalah ini karena media dokumen yang biasanya besar diikuti dengan kata kunci yang secara umumnya tidak panjang. Kedua hal itu mendukung keunggulan dari algoritma KMP yang berkerja baik pada media pencarian yang besar dan kata kunci yang pendek.

VI. UCAPAN TERIMA KASIH

Pertama-tama penulis ingin mengucapkan puji dan syukur kepada Tuhan Yang Maha Esa atas anugerah dan bimbingan-Nya penulis dapat menyelesaikan makalah berjudul "Aplikasi *Knuth-Morris-Pratt Algorithm* pada Pencarian Isi Dokumen Lokal" ini dengan baik dan lancar. Selain itu, penulis juga ingin mengucapkan terima kasih kepada Dosen pengajar Dr. Nur Ulfa Maulidevi, S.T., M.Sc, Dr. Masayu Leylia Khodra, S.T., M.T. dan Dosen pembimbing Dr. Ir. Rinaldi Munir, M.T., atas petunjuk dan arahan yang telah diberikan selama proses pengerjaan makalah ini berlangsung. Penulis juga tak lupa untuk berterima kasih kepada teman-teman yang memberikan masukan dan kritik selama penulis menyelesaikan makalah ini.

REFERENSI

[1] Munir, Rinaldi. 2018. *Strategi Algoritma*. Bandung: Penerbit Informatika.

[2] Apache. 2018. *Apache POI* [dokumentasi]. Tersedia dalam:https://poi.apache.org/document/index.html. [diakses 11 Mei 2018 pukul 09.12 WIB].

[3] Cormen, Thomas. 2001. *Introduction to Algorithms* [edisi 2]. United States: MIT Press dan McGraf-Hill.

[4] Crochemore, Maxime. 2003. *Jewels of stringology*. River Edge: World Scientific.

[5] Szpankowski, Wojciech. 2001. *Average case analysis of algorithms on sequences*. Chichester: Wiley.

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, 12 Mei 2018

Seperayo - 13516068