Tugas Kecil 4

IF2211 Strategi Algoritma

Ekstraksi Informasi dari Artikel Berita dengan Algoritma Pencocokan String

Disusun oleh

Indra Febrio Nugroho 13518016

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2020

BAB I
TEORI SINGKAT

1.1 Algoritma Knuth-Morris-Pratt

Algoritma Knuth-Morris-Pratt merupakan salah satu algoritma pencarian string. Algoritma ini juga dikenal sebagai algoritma KMP (singkatannya), dikembangkan secara terpisah oleh Donald Knuth dan Vaughan Pratt pada tahun 1970, serta secara independen oleh James H. Morris. Ketiga penemu tersebut mempublikasikan algoritma ini pada tahun 1977. Dari nama algoritma ini kita dapat melihat dibuat berdasarkan tiga nama penemunya.

Algoritma Knuth-Morris-Pratt adalah algoritma yang mencari pattern dalam sebuah text dari kiri ke kanan seperti algoritma Brute Force. Algoritma ini melakukan pergeseran pattern lebih "cerdas" dibandingkan dengan brute force.

Secara sistematis, langkah-langkah yang dilakukan algoritma Knuth-Morris-Pratt pada saat mencocokkan string:

1. String pattern (kata yang dicari) akan dipecah menjadi array karakter. String text (teks, artikel) akan dipecah menjadi array karakter
2. Algoritma Knuth-Morris-Pratt mulai mencocokkan pattern pada awal teks.
3. Dari kiri ke kanan, algoritma ini akan mencocokkan karakter per karakter pattern dengan karakter di teks yang bersesuaian, sampai salah satu kondisi berikut dipenuhi:
    A. Karakter di pattern dan di teks yang dibandingkan tidak cocok (mismatch).
    B. Semua karakter di pattern cocok. Kemudian algoritma akan memberitahukan penemuan di posisi ini.
4. Algoritma kemudian menggeser pattern berdasarkan tabel next, lalu mengulangi langkah 2 sampai pattern berada di ujung teks.

Kelebihan dari algoritma Knuth-Morris-Pratt selain cepat juga sangat baik digunakan pada file berukuran besar karena pencarian kecocokan tidak perlu kembali ke belakang pada input teks.

Namun algoritma ini memilki kekurangan yakni efektifitas dari algoritma ini akan berkurang seiring dengan bertambahnya jumlah jenis karakter dari teks.

1.2 Algoritma Boyer-Moore

Algoritme Boyer-Moore adalah salah satu algoritme pencarian string, dipublikasikan oleh Robert S. Boyer, dan J. Strother Moore pada tahun 1977.
Algoritme ini dianggap sebagai algoritme yang paling efisien pada aplikasi umum. Tidak seperti algoritme pencarian string yang ditemukan sebelumnya, algoritme Boyer-Moore mulai mencocokkan karakter dari sebelah kanan pattern. Ide di balik algoritme ini adalah bahwa dengan memulai pencocokan karakter dari kanan, dan bukan dari kiri, maka akan lebih banyak informasi yang didapat

Secara sistematis, langkah-langkah yang dilakukan algoritme Boyer-Moore pada saat mencocokkan string adalah:

1. Algoritme Boyer-Moore mulai mencocokkan pattern pada awal teks.
2. Dari kanan ke kiri, algoritme ini akan mencocokkan karakter per karakter pattern dengan karakter di teks yang bersesuaian, sampai salah satu kondisi berikut dipenuhi:
    1. Karakter di pattern dan di teks yang dibandingkan tidak cocok (mismatch).
    2. Semua karakter di pattern cocok. Kemudian algoritme akan memberitahukan penemuan di posisi ini.
3. Algoritme kemudian menggeser pattern dengan memaksimalkan nilai penggeseran good-suffix dan penggeseran bad-character, lalu mengulangi langkah 2 sampai pattern berada di ujung teks.

1.3 Regex

Regular Expression adalah Sebuah cara mendefinisikan language yang lebih tepat dibandingkan dengan menggunakan cara ellipsis (diakhiri dengan ...)

Dengan memanfaatkan closure, bila S = { x } maka L4 = S* Dapat juga ditulis sebagai L4 = {x}*

Kleene Star tidak hanya dapat diaplikasikan untuk set namun juga langsung ke alphabet. Contoh: x* Ekpresi sederhana x* akan dipakai untuk mengekspresikan pengulangan dari x (bisa juga tidak sama sekali)

BAB II
KODE PROGRAM

1. KMP.py

```python
# 13518016
# Indra Febrio Nugroho
# Knuth-Morris-Pratt Algorithm

def KMPMatch(text, pattern):
    m = len(pattern)
    n = len(text)

    # create container that will hold the longest prefix suffix values for pattern
    lps = [0 for i in range(m)]
    # calculate the lps
    calculateLPS(pattern, lps)

    total = 0 # total matched pattern in the text
    matchedIdx = [] # container for index of matched pattern in the text
    i = 0 # index for text
    j = 0 # index for pattern
    while i < n:
        if (pattern[j] == text[i]):
            i += 1
            j += 1
            if (j == m):
                return (i-j)
                matchedIdx.append(i-j)
                total += 1
                j = lps[j-1]
        elif (j > 0):
            j = lps[j-1]
        else:
            i += 1

    # if (total > 0) :
        # print("Total matched pattern in the text: ", str(total))
        # print("Matched at index: ", end="")
        # print(matchedIdx)
    if (total == 0):
        # print("Pattern not found")
        return (-1)
```

```python
def calculateLPS(pattern, lps):
    m = len(pattern)

    i = 1
    j = 0 # length of the previous longest prefix suffix

    while (i < m):
        if (pattern[i]== pattern[j]):
            lps[i] = j + 1
            i += 1
            j += 1
        elif (j > 0):
            j = lps[j-1]
        else:
            lps[i] = 0
            i += 1
```

2. BM.py

```python
def BMMatch(text, pattern):
    m = len(pattern)
    n = len(text)

    i = m - 1
    total = 0 # total matched pattern in the text
    matchedIdx = []

    lastOcc = buildLastOccurence(pattern)

    if (i > n - 1) :
        return (-1)
    else :
        j = m - 1
        while (i <= n - 1):
            if (pattern[j] == text[i]):
                i -= 1
                j -= 1
                if (j == -1) :
                    return (i+1)
                    total += 1
                    matchedIdx.append(i+1)
                    j = m - 1
                    i += 2*m
            else :
                lo = lastOcc[ord(text[i])]
                i += m - min(j, 1+lo)
                j = m - 1

    if (total == 0):
        return (-1)
```

```python
def buildLastOccurence(pattern):
    # length of pattern
    m = len(pattern)
    # Init all occurrence as -1
    lastOcc = [-1 for i in range(128)]

    # Fill the value of last occurrence to the array
    for i in range(m):
        lastOcc[ord(pattern[i])] = i;

    return lastOcc
```

3. Extractor.py

```python
# 13518016
# Indra Febrio Nugroho
# Regex and Extractor

import re
import copy
from nltk import sent_tokenize
from BM import BMMatch
from KMP import KMPMatch

def RegexMatch(text, pattern):
    if (re.search(pattern, text)):
        return 1
    else:
        return -1

def algChooser(alg, text, keyword):
    idx = -1
    if (alg == "kmp"):
        idx = KMPMatch(text.lower(), keyword.lower())
    elif (alg == 'bm'):
        idx = BMMatch(text.lower(), keyword.lower())
    else:
        idx = RegexMatch(text.lower(), keyword.lower())
    return idx
```

```python
def processFile(path, files, keyword, alg):
    listOfTotal = []
    listOfTime = []
    listOfWords = []
    listOfFiles = []
    for f in files:
        if (f.endswith(".txt")):
            file = open(path + f, 'r')
            text = file.read()
            reTime = re.compile(r'([A-Za-z]{4,6}\,? )?\(?\d{1,2}[\/\- ](\d{1,2}|[a-zA-Z]{3,})[\/\- ]?\d{2,4}\)?( (pukul )?\d{1,2}[\.\:]\d{1,2} ((WITA)|(WI
            timeG = reTime.search(text)
            globalTime = timeG.group(0)
            arrOfWords = sent_tokenize(text)

            for words in arrOfWords:
                idx = algChooser(alg, words, keyword)
                if (idx > 0):
                    reTotal = re.compile(r'(\d{1,3}(\.|\,))*\d{1,} (([Oo]rang)|([Pp]asien)|([OoPp]dp)|([Kk]asus))')
                    totalRes = reTotal.search(words)
                    reTime = re.compile(r'([A-Za-z]{4,6}\,? )?\(?\d{1,2}[\/\- ](\d{1,2}|[a-zA-Z]{3,})[\/\- ]?\d{2,4}\)?( (pukul )?\d{1,2}[\.\:]\d{1,2} ((W
                    timeRes = reTime.search(words)

                    if (totalRes and timeRes):
                        w = totalRes.group().split(' ')
                        listOfTotal.append(copy.deepcopy(w[0]))
                        listOfTime.append(copy.deepcopy(timeRes.group()))
                        listOfWords.append(copy.deepcopy(words))
                        listOfFiles.append(copy.deepcopy(f))

                    elif (totalRes):
                        w = totalRes.group().split(' ')
                        listOfTotal.append(copy.deepcopy(w[0]))
                        listOfTime.append(copy.deepcopy(globalTime))
                        listOfWords.append(copy.deepcopy(words))
                        listOfFiles.append(copy.deepcopy(f))

    return listOfTotal, listOfTime, listOfWords, listOfFiles
```

4. App.py

```python
from flask import Flask, redirect, url_for, render_template, request
from werkzeug.utils import secure_filename
from Extractor import processFile

app = Flask(__name__)

@app.route("/", methods=['GET', 'POST'])
def homePage():
    if (request.method == 'POST'):
        path = "../test/"
        fileFromWeb = request.files.getlist('file[]')
        files = []
        for f in fileFromWeb:
            files.append(f.filename)
        keyword = request.form['kw']
        algorithm = request.form['algo']
        listOfTotal, listOfTime, listOfWords, listOfFiles = processFile(path, files, keyword, algorithm)
        length = len(listOfTotal)
        return render_template('app.html', length=length, keyword=keyword, total=listOfTotal, time=listOfTime, words=listOfWords, files=listOfFiles)
    else:
        keyword = ""
        listOfTotal = []
        listOfTime = []
        listOfWords = []
        listOfFiles = []
        length = len(listOfTotal)
        return render_template('app.html', length=length, keyword=keyword, total=listOfTotal, time=listOfTime, words=listOfWords, files=listOfFiles)

if __name__ == "__main__":
    app.run(debug=True)
```

5. app.html

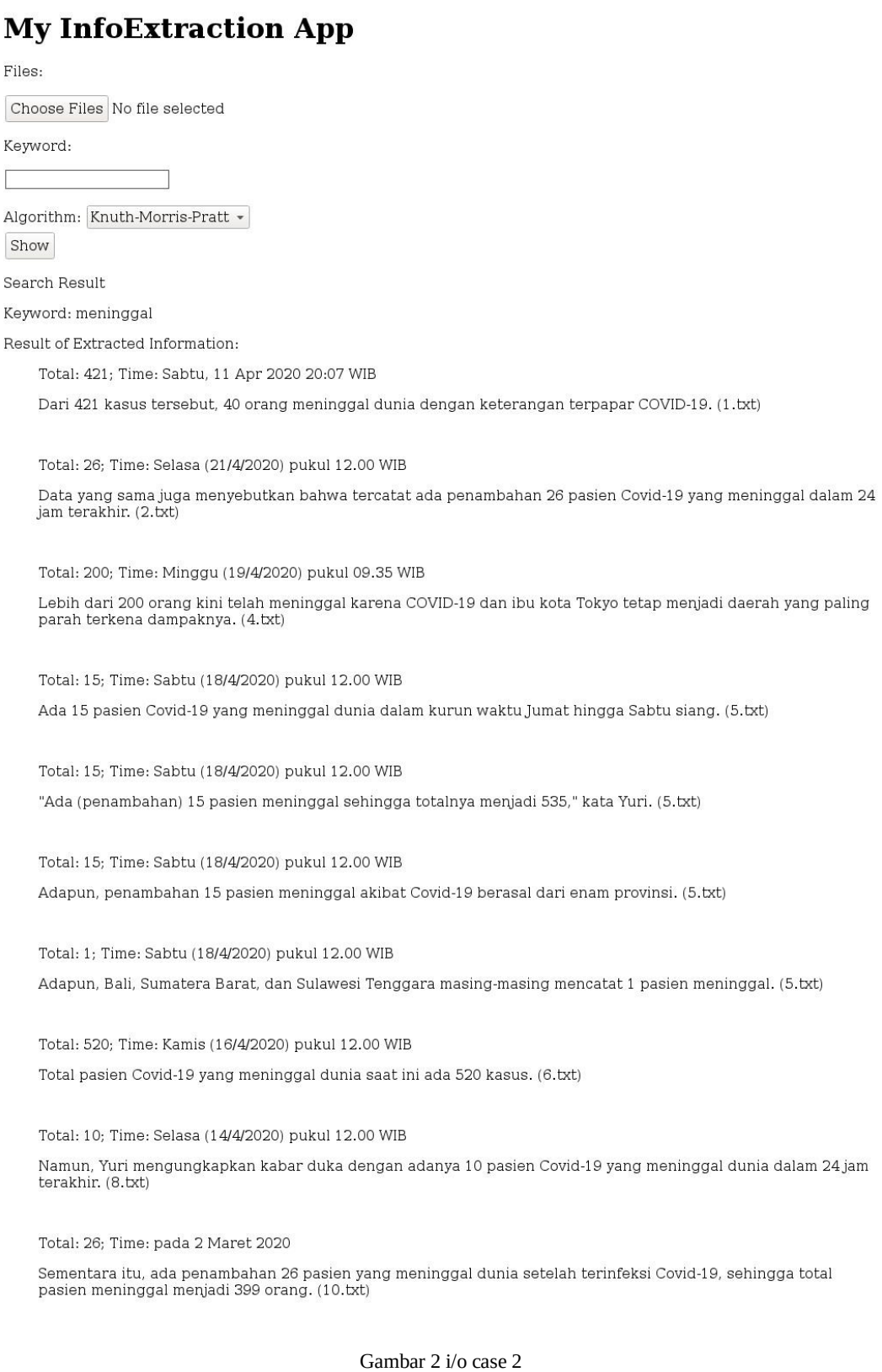
```
{% extends "layout.html" %}
{% block body %}
<!DOCTYPE html>
<h1>My InfoExtraction App</h1>

<form name="allForm" class="form-inline" method="POST" action="/" onsubmit="/" enctype="multipart/form-data">
  <div class="form-group">
    <div class="input-group">
        <p>Files:</p>
        <p><input type="file" name="file[]" multiple=""></p>
        <p>Keyword:</p>
        <p><input type="text" name="kw" /></p>
        <span class="input-group-addon">Algorithm:</span>
            <select name="algo" class="selectpicker form-control">
                <option value="kmp">Knuth-Morris-Pratt</option>
                <option value="bm">Boyer Moore</option>
                <option value="regex">Regex</option>
            </select>
    </div>
    <button type="submit" class="btn btn-default">Show</button>
  </div>
</form>

<p>Search Result</p>

<p>Keyword: {{keyword}}</p>
<p>Result of Extracted Information:</p>

<ol>
{%for i in range(0, length)%}
    <p>Total: {{total[i]}}; Time: {{time[i]}}</p>
    <p>{{words[i]}} ({{files[i]}})</p>
    <br>
{%endfor%}

</ol>


{% endblock %}
```

BAB III
SCREENSHOT I/O PROGRAM

3.1 Input Output Program Case 1 (“positif”, KMP)

Gambar 1 i/o case 1

3.2 Input Output Program Case 2 (“meninggal”, BM)



Gambar 2 i/o case 2

3.3 Input Output Program Case 3 (“pdp”, Regex)

Gambar 3 i/o case 3

3.4 Kesimpulan

Luaran dari tugas ini dapat dimuat secara ringkas dalam tabel berikut

Tabel 1 Output Persoalan

Poin	Ya	Tidak
1. Program berhasil dikompilasi	v	
2. Program berhasil *running*	v	
3. Program dapat menerima input dan menuliskan output	v	
4. Luaran sudah benar untuk semua data uji	v	

REFERENSI

Munir, Rinaldi. 2004. *Diktat Bahan Kuliah Strategi Algoritmik.* Bandung: Departemen Teknik Informatika, Institut Teknologi Bandung.

https://unfinishedcreativework.wordpress.com/2017/06/10/algoritma-knuth-morris-pratt/ diakses pada 22 April 2020, 02:33 WIB

Lecroq, Thierry Charras, Christian. 2001. Handbook of Exact String Matching Algorithm.

Boyer, Robert Moore, J. 1977. A Fast String Searching Algorithm. Comm. ACM 20: 762–772