**Laporan Tugas Kecil**

**IF2211 - Strategi Algoritma**

**Ekstraksi Informasi dari Artikel Berita dengan Algoritma Pencocokan String**



Disusun oleh:

Gregorius Jovan Kresnadi

13518135

**SEKOLAH TINGGI ELEKTRO DAN INFORMATIKA**

**TEKNIK INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2020**

**Bab I**

**Algoritma Pencocokan String**

Algoritma pencocokan string atau algoritma *string matching* adalah sebuah algoritma yang digunakan untuk menemukan sebuah pola dalam string. Pada umumnya, dalam algoritma dibutuh dua buah nilai. Nilai T atau teks adalah sebuah string yang panjangnya *n* buah karakter. Nilai P atau pattern adalah sebuah string yang panjangnya *m* buah karakter yang akan dicari dalam string T. Terdapat beberapa algoritma string matching yang dapat digunakan, yaitu algoritma Brute Force, algoritma Knuth-Morris-Pratt (KMP), algoritma Boyer-Moore, dan Regular Expression atau ekspresi regular. Dalam tugas ini, algoritma Brute Force tidak diimplementasikan.

1. **Algoritma Knuth-Morris-Pratt (KMP)**

Algoritma *Knuth-Morris-Pratt* (KMP) adalah algoritma string matching yang mencocokan pattern dari kiri ke kanan. Algoritma ini melakukan penggeseran dalam mencari pola dengan cerdas. Dalam pencocokan, pola akan disejajarkan dulu dari karakter pertama T, dan mencocokkannya satu per satu ke kanan, namun jika terdapat perbedaan, T[i] ≠ P[j] maka algoritma akan menggeser pola sebanyak jumlah prefix terbesar dari P[0..j-1] yang merupakan suffix P[1..j-1]. Untuk melakukan in, algoritma KMP menggunakan sebuah fungsi preprocessing yang dinamakan KMP Border Function. Fungsi ini akan membaca pola yang akan dicocokan untuk mencari prefix yang bisa cocok dengan suffix pola itu sendiri.

Keuntungan memakai algoritma KMP adalah pengecekan tidak pernah dilakukan berulang, atau bergerak mundur, ini membuat algoritma sangat baik dalam memerika file-file besar. Namun semakin banyak variasi karakter yang digunakan, algoritma berpotensi bekerja semakin lama karena frekuensi ketidakcocokannya pun naik. Selain itu, algoritma dasarnya tidak mencatat karakter jenis apa yang dibaca, hanya membandingkannya.

1. **Algoritma Boyer-Moore (BM)**

Algoritma *Boyer-Moore* (BM) adalah algoritma string matching yang menggunakan 2 buah teknik, yaitu teknik *looking-glass* dan teknik *character-jump*. Teknik *looking-glass* adalah saat algoritma membandingkan pola dengan teks dari arah kanan ke kiri. Namun, hanya pola yang dibaca terbalik, penempatan pola pada teks dan penggeseran tetap berlangsung dari kiri ke kanan. Kemudian, teknik *character-jump* adalah saat algoritma menemukan ketidakcocokan pada pola dan teks, misalnya saat T[i] ≠ P[j], sehingga ada 3 kasus yang dapat terjadi. Kasus pertama, saat jika P mengandung T[i] pada pola, maka P akan digeser ke kanan hingga posisi T[i] sejajar dengan P dengan karakter tersebut. Kasus kedua, jika P mengandung T[i] namun sudah pernah terlewati, maka P di geser ke kanan sekali menjadi sejajar denngan T[i+1]. Kasus ketiga terjadi saat T[i] tidak terdapat pada P, dan P digeser ke kanan hingga T[i] tidak sejajar lagi dengan pola.

Untuk menggunakan teknik *character-jump,* algorimta BM juga mempunyai sebuah fungsi preprocessing yang melakukan mapping seluruh karakter dan indexnya dimana terakhir muncul pada pola ke sebuah larik/array. Karakter yang dipetakan adalah seluruh karakter yang mungkin muncul pada teks. Jika karakter tidak muncul pada pola, maka index yang disimpan adalah -1. Dengan teknik ini, algoritma dapat, melakukan string matching secara cepat pada teks bahasa Inggris atau Indonesia, namun sangat lambat pada sebuah string binary karena variasinya yang sangat sedikit.

1. **Regular Expression**

Regular Expression (Regex) adalah sebuah notasi standar yang mendeskripsikan suatu pola berupa urutan karater atau string. Regex dapat digunakan pada string matching dengan sangat efisien, selain itu Regex juga adalah sebuah ilmu dasar yang umum perlu dikuasai oleh programmer, terutama yang fokus di bidang Natural Language Processing.

Regex memiliki 2 jenis notasi, yaitu literal characters dan metacharacters. Literal characters adalah karakter-karakter umum seperti alfabet dan angka, namun metacharacters adalah karakter khusus yang membuat Regex unggul dalam mencari pola. Contohnya, karakter titik “.” dapat dianggap sebagai karakter apa saja, jadi sebuah regex dengan notasi .andi dapat mengembalikan huruf Handi, Tandi, atau bandi.

Sumber: http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/stima19-20.htm

**Bab II**

**Kode Program**

Berikut adalah kode program inti yang digunakan untuk menjalankan program

app.py

|  |
| --- |
| from flask import Flask, render\_template, url\_for, request, redirect  from os import path  from Form import Form  app = Flask(\_\_name\_\_)  app.config["FILE\_UPLOADS"] = path.join(app.root\_path, 'static', 'files')  def allowed\_file(filename):      if not "." in filename:          return False      ext = filename.rsplit(".", 1)[1]      if ext.upper in app.config["ALLOWED\_EXT"]:          return True      else:          return False  @app.route('/', methods=['POST', 'GET'])  def index():      keyword = ""      if request.method == 'POST':          answers = []          keyword = request.form['keywordInput']          option = request.form['radioMethod']          if request.files:              files = request.files.getlist('sourceFile')              for file in files:                  if file.filename == "" or file.filename == " " or file.filename == None:                      print("NAMELESS FILE")                      redirect(request.url)                  try:                      file.save(path.join(app.config["FILE\_UPLOADS"], file.filename))                      print("File saved")                  except Exception as e:                      print(e)                      print("NO FILES UPLOADED")                  form = Form(file.filename, keyword, option)                  form.readFile()                  answers.append(form)                try:              return render\_template('index.html', keyword=keyword, answers=answers)              return redirect(request.url)          except Exception as e:              return "There was an issue processing the files"              return render\_template('index.html', keyword=keyword, answers=answers)        else:          return render\_template("index.html", keyword = keyword)  def postAnswers(ans):      pass  if \_\_name\_\_ == "\_\_main\_\_":      app.run(debug=True) |

Matcher.py

|  |
| --- |
| """ This module creates an matcher to indicate where      {pattern} occurs in {text}  """  import re  class Matcher:      """Class for matching pattern on a given text"""      def \_\_init\_\_(self, text, pattern):          'Matcher ctor'          self.text = text          self.pattern = pattern      def setText(self, text):          'Sets the text to read'          self.text = text      def setPattern(self, pattern):          'Sets the pattern to match'          self.patterm = pattern      def BMMatch(self):          'String matcher using Boyer-Moore Algorithm'          last = self.computeLastOccurence();          n = len(self.text)          m = len(self.pattern)          i = m-1          if i > n-1:              return -1   # no match if pattern longer than text          j = m-1          while 1:              if self.pattern[j] == self.text[i]:                  if j == 0:                      return i    # found match                  else:                      i -= 1                      j -= 1              else:                  lastOcc = last[ord(self.text[i])]   # last occurance                  i = i + m - min([j, 1+lastOcc])                  j = m - 1              if i > n-1:                  break          return -1       # no match      def computeLastOccurence(self):          'Returns array storing index of last occurence of each ASCII character in pattern'          last = [-1 for x in range(256)]          for i in range (len(self.pattern)):              last[ord(self.pattern[i])] = i          return last      def KMPMatch(self):          'String matcher using Knuth-Morris-Pratt Algorithm'          n = len(self.text)          m = len(self.pattern)          fail = self.computeFail()          i = 0          j = 0          while i < n:              if (self.pattern[j] == self.text[i]): # match                  if j == m-1:                      return i-m+1                  i += 1                  j += 1              elif j > 0:                  j = fail[j-1]              else:                  i += 1          return -1 # no match      def computeFail(self):          'Returns array storing size where repeating patterns happen'          fail = [0 for i in range (len(self.pattern))]          m = len(self.pattern)          j = 0          i = 1          while i < m:              if self.pattern[j] == self.pattern[i]: # j+1 chars match                  fail[i] = j+1                  i += 1                  j += 1              elif j>0:   # j follows matching prefix                  j = fail[j-1]              else:       # no match                  fail[i] = 0                  i += 1          return fail      def REMatch(self):          'String matcher using Regular Expression'          pa = self.pattern          patt = re.compile(self.pattern)          matches = patt.search(self.text) == None          if matches:              return -1          else:              return 1 |

Extractor.py

|  |
| --- |
| """  This module loads file and extracts sentences  """  from nltk.tokenize import sent\_tokenize, word\_tokenize  from Matcher import Matcher  import re  def findAmount(sent):      'Finds an essential amount in a sentence'      toRead = sent.lower()      patt = re.compile(r'(\d\*\.?\,?\d+)\s(orang?|kasus?|jiwa?|tewas?|meninggal?|korban?|)')      matches = patt.finditer(toRead)      return matches  def findDate(sent):      'Finds the date in a sentence'      min = 99999      max = 0      patt = re.compile(r'\(+[1-3]?[0-9]\/[1]?[0-9]/\d\d\d\d\)+|\(\d+\/\d+\)|Senin|Selasa|Rabu|Kamis|Jumat|Sabtu|Minggu|\d\d\:\d\d|WIB|WITA|WIT|kemarin|minggu')      matches = patt.finditer(sent)      for x in matches:          if x.start() < min:              min = x.start()          if x.end() > max:              max = x.end()      return sent[min:max]  def formatInfo(article, amount, date):      'Formats the string'      form = ("""  Jumlah: {}  Waktu: {}  {}""")      return form.format(amount, date, article)  class Extractor:      'Class for extraction information from text, usually amount and time, based on keyword'      def \_\_init\_\_(self, text, pattern, method):          'Extractor ctor'          self.text = text          self.keyword = pattern          self.method = method      def findInfoWithMethod(self):          'Finds and collects all needed information'          result = []          list\_of\_sentence = sent\_tokenize(self.text)          for sent in list\_of\_sentence:              res = Matcher(sent.lower(), self.keyword)              articleDate = ""              if self.method == 'optionBM':                  r = res.BMMatch()              elif self.method == 'optionKMP':                  r = res.KMPMatch()              elif self.method == 'optionRE':                  r = res.REMatch()              if r > -1:                  result.append(sent)              if articleDate == "" or articleDate == " ":                  resDate = Matcher(sent, self.keyword)                  # articleDate = resDate          return result, articleDate      def extractInformation(self):          'Extracts the information from the file and returns as a formatted string'          result, articleDate = self.findInfoWithMethod()          forms = []          for res in result:              date = findDate(res)              amount = []              for a in findAmount(res):                  amount.append(a.group())              if date == " ":                  date = articleDate              forms.append(formatInfo(res, amount, date))          return forms |

Form.py

|  |
| --- |
| """  This module loads file and extracts sentences  """  from nltk.tokenize import sent\_tokenize, word\_tokenize  from Matcher import Matcher  from os import path  from Extractor import Extractor  import re  class Form:      'Class for forming results based on folder input'      def \_\_init\_\_(self, filename, keyword, method):          'Form ctor'          self.filename = filename          self.article = ""          self.info = []          self.keyword = keyword.lower()          self.method = method      def readFile(self):          'Reads file from filename and puts results in info'          file = open(path.join('static', 'files', self.filename));          self.article = file.read()          extract = Extractor(self.article, self.keyword, self.method)          self.info = extract.extractInformation() |

Base.html

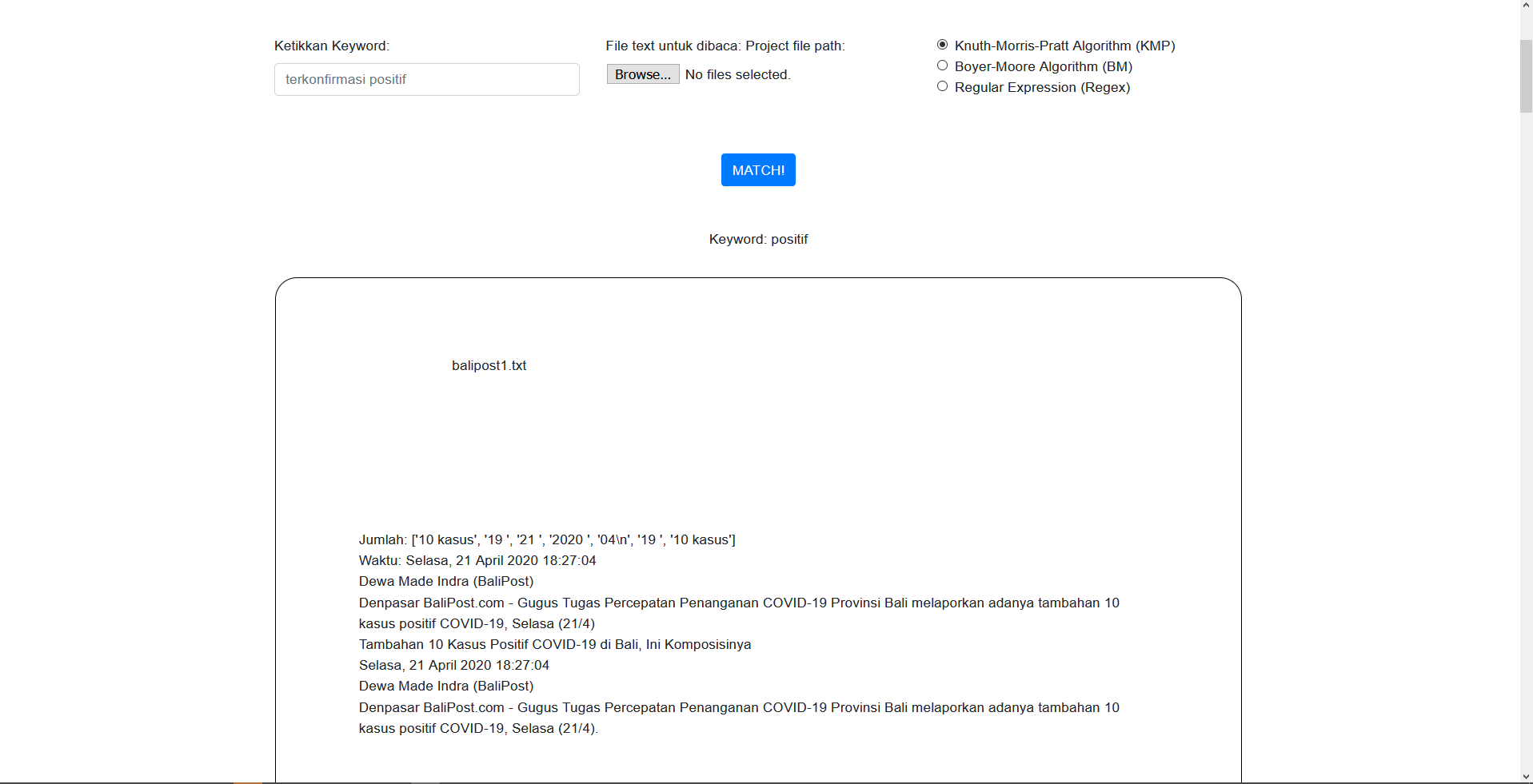
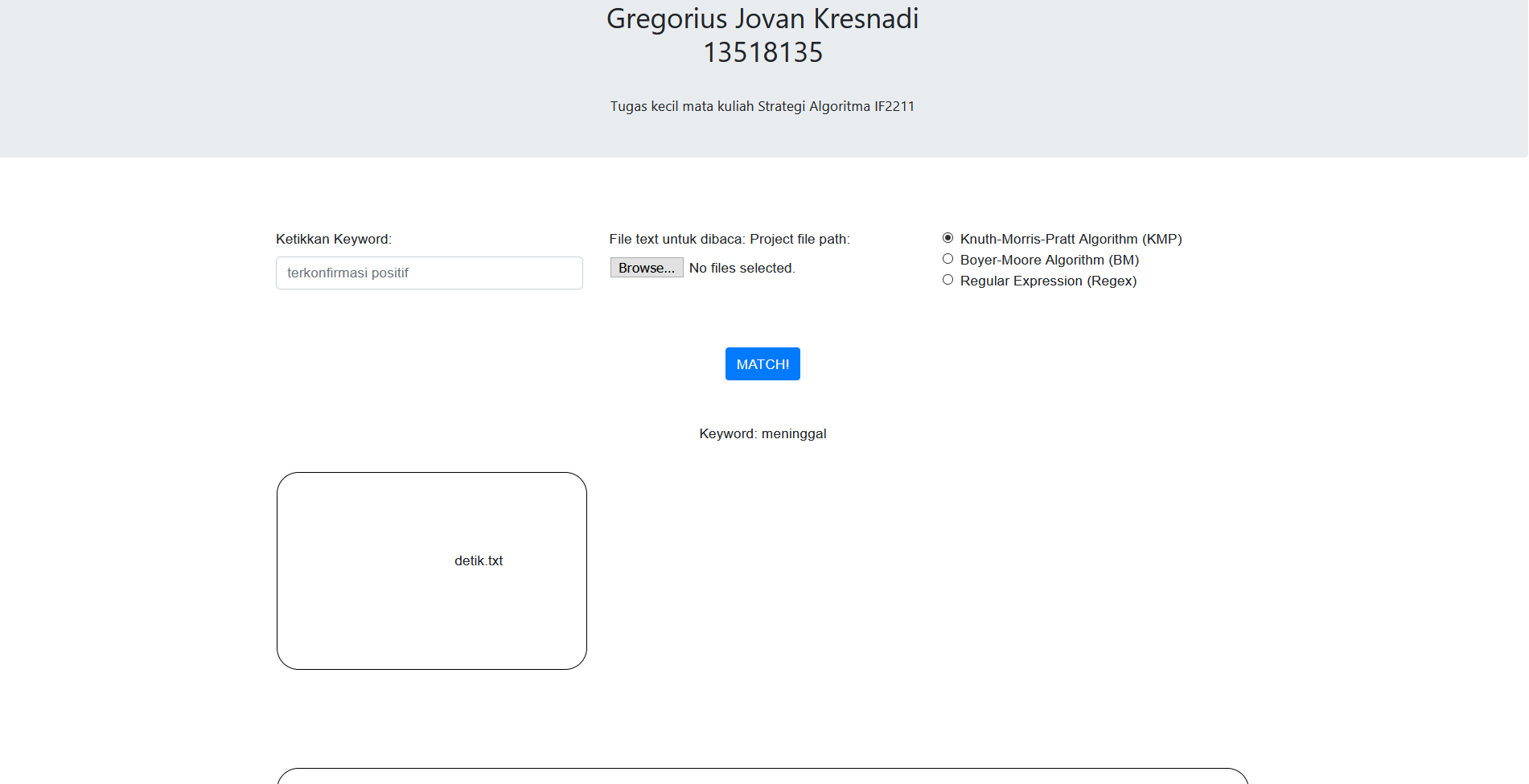
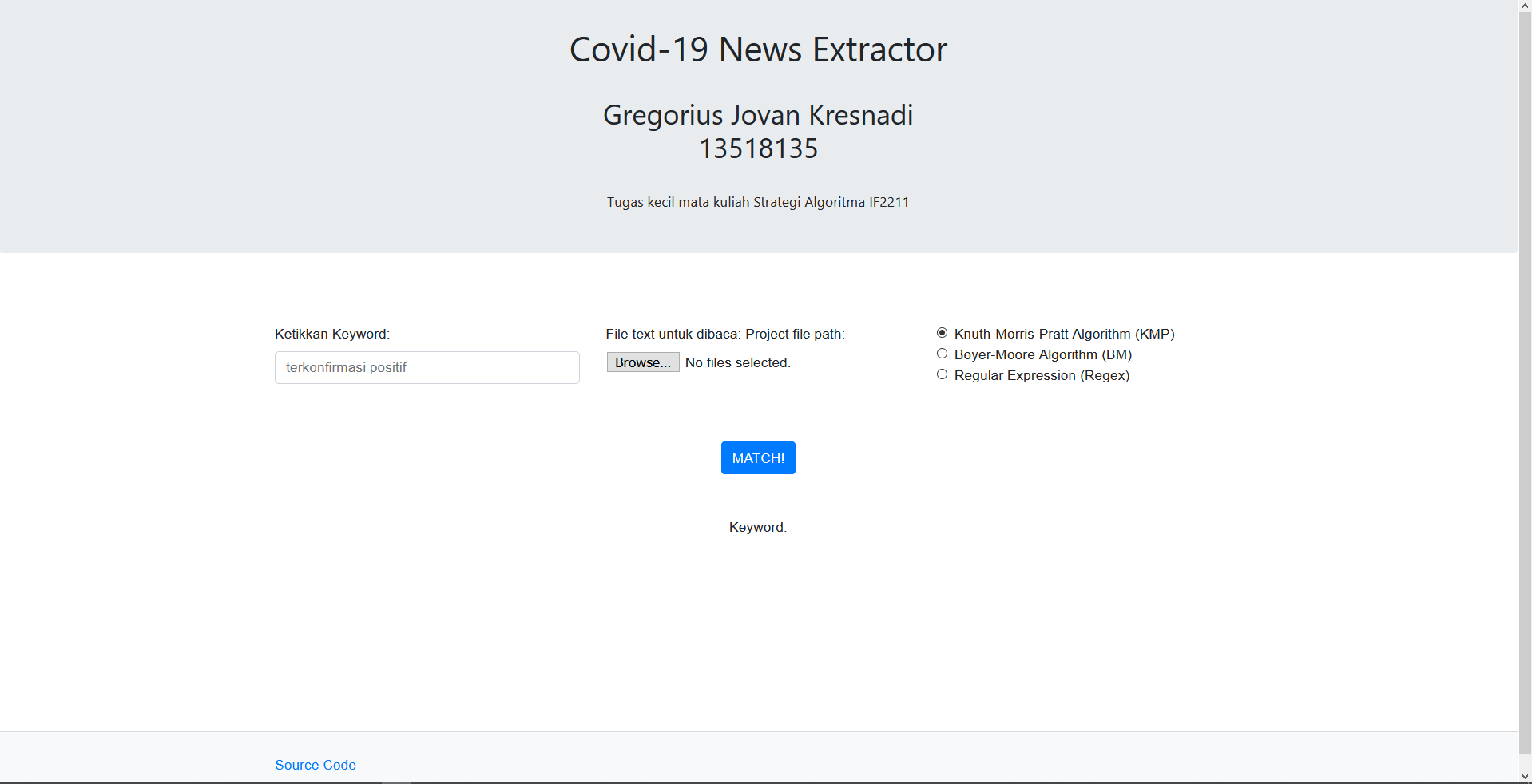
|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <link rel="stylesheet" href="{{ url\_for('static', filename='css/main.css' ) }}">      <link rel="stylesheet" href="{{ url\_for('static', filename='css/sticky-footer.css' ) }}">      <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css">      <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>      <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>      <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js"></script>      {% block head %}{% endblock %}  </head>  <body>      {% block body %}{% endblock %}  </body>  </html> |

Index.html

|  |
| --- |
| {% extends  "base.html" %}  {% block head %}  <title>      Covid-19 News Extracter  </title>  {% endblock %}  {% block body %}  <div class="jumbotron text-center" style="padding: 2rem">      <h1>Covid-19 News Extractor</h1>      <br>      <h2>Gregorius Jovan Kresnadi<br>13518135</h2>      <br>      <p>Tugas kecil mata kuliah Strategi Algoritma IF2211</p>  </div>  <div class="container">      <form action="/" method="POST" enctype="multipart/form-data" >          <div class="row">              <div class="col-md-4">                  <div class="form-group">                      <label for="keywordInput">Ketikkan Keyword:</label>                      <input type="text" class="form-control" name="keywordInput" id="keywordInput" placeholder="terkonfirmasi positif">                  </div>              </div>              <div class="col-md-4">                  <div class="form-group">                      <label for="sourceFileControl">File text untuk dibaca:</label>                      Project file path: <input type="file" class="form-control-file" name="sourceFile"  id="sourceFile" multiple="">                  </div>              </div>              <div class="col-md-4">                  <div class="form-group">                      <div class="form-check">                          <input class="form-check-input" type="radio" name="radioMethod" id="radioKMP" value="optionKMP" checked>                          <label class="form-check-label" for="radioKMP">                          Knuth-Morris-Pratt Algorithm (KMP)                          </label>                      </div>                      <div class="form-check">                          <input class="form-check-input" type="radio" name="radioMethod" id="radioBM" value="optionBM">                          <label class="form-check-label" for="radioBM">                          Boyer-Moore Algorithm (BM)                          </label>                      </div>                      <div class="form-check">                          <input class="form-check-input" type="radio" name="radioMethod" id="radioRegex" value="optionRegex">                          <label class="form-check-label" for="radioRegex">                          Regular Expression (Regex)                          </label>                      </div>                  </div>              </div>          </div>          <div class="row match">              <input class="btn btn-primary" type="submit" value="MATCH!">          </div>          <div class="row">              <div class="container keyword" >                  Keyword: {{ keyword }}              </div>              {% for ans in answers %}              <div class="row-answers">                  <div class="ans-container">                      <div class="container">                          {{ ans.filename }}                      </div>                      {% for info in ans.info %}                      <div class="container info">                          {{ info }}                      </div>                      <hr>                      {% endfor %}                  </div>              </div>              {% endfor %}          </div>      </form>  </div>  <div class="footer">      <footer class="card-footer bg-light footer-font-style">          <div class="container">              <a href="https://github.com/BaconLover307/Covid19-news-extractor"> Source Code </h3>          </div>      </footer>  </div>  {% endblock %} |

**Bab III**

**Screenshot Program**



**Tabel Poin**

|  |  |  |
| --- | --- | --- |
| Poin | Ya | Tidak |
| 1. Program berhasil dikompilasi | √ |  |
| 1. Program berhasil *running* | √ |  |
| 1. Program dapat menerima input dan menuliskan output | √ |  |
| 1. Luaran sudah benar untuk semua data uji |  | √ |