**LAPORAN TUGAS KECIL 4**

**STRATEGI ALGORITMA**

**KELAS 02**

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

Oleh :

Kevin Austin Stefano (13518104)

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA** **INSTITUT TEKNOLOGI BANDUNG  
2020**

**DASAR TEORI**

**Pencocokan string Knuth-Morris-Pratt (KMP)**

Algoritma pencarian string ini dikembangkan oleh Donald E. Knuth pada tahun 1967. Algoritma pencarian string Knuth-Mooris-Pratt memiliki langkah kerja yang meningkatkan pergeseran dilakukan sehingga jauh lebih efisien dibandingkan dengan metode *brute force.* Ide umum dari algoritma KMP ini adalah dengan melakukan pergeseran karakter dari kiri ke kanan. Saat terjadinya *mismatch* antara text(T) pada T[i] denggan pattern (P) pada P[j] atau dengan kata lain T[i] != P[j], maka langkah yang kita lakukan adalah dengan menggeser pattern sedemikian rupa sehingga *largest prefix* Pattern[0..j-1] adalah sama dengan *suffix pattern*  dari P[1..j-1] dan sejajar dengan *text* bersangkutan.

**Pencocokkan String Boyer-Moore**

Algoritma Booyer-Moore adalah algoritma pencarian string yang dikembangkan oleh Robert S. Boyer dan J. Strother Moore pada tahun 1977. Algoritma ini adalah algoritma pencaria string yang paling efisien dibandingkan algoritma-algoritma lainnya Konsep penerapannya algoritma ini adalah sebagai berikut, bahwa kita melakukan pencocokan dari kanan ke kiri sehingga lebih efisien. Algoritma ini pada umumnya melakukan 2 teknik, yaitu melakukannya dengan *looking-glass* teknik dimana memeriksa kecocokan pattern yang dimulai daari indeks terakhir P. Dan selanjutnya kita bisa melakukannya dengan teknik *character jump* dimana terdapat *mismatch* maka kita melakukan *jump* terhadap indeks text sehingga lebih efisien.

**Pencocokkan String Regex.**

Algoritma regex adalah algoritma yang dilakukan berdasarkan teori otomata bahasa formal. Algoritma ini terdiri dari kumpulan konstanta dan operator dan melakukan pencocokan terhadap pola yang dicari berdasarkan urutan karakter atau string. Regex akan melakukan pencarian string secara literal dengan pembatasan yang didefinisikan melalui quantifier maupun boundary matchersnya. Quantifier dalam regex digunakan untuk mendefinisikan jumlah pengulangan pola. Sedangkan Boundary Matcher digunakan untuk mencari pola yang yang ada pada suatu posisi tertentu.

**ANALISIS ALGORITMA**

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

1. Algoritm KMP memulai pencocokan string dari indeks awal *teks* T[0] dan *pattern* P[0]
2. Selama belom ditemukan atau indeks *teks(*i) belom lebih dari panjang *teks*(len(i)), maka
   1. Jika teks T[i] sama dengan pattern P[j], maka i digeser satu ke kenan begitu juga j digeser satu ke kanan (i= i+1, j= j+1)
   2. Jika terjadi *mismatch* di indeks j >0*¸* maka kita memulai mencari Pattern dari *largest prefix* Pattern [0..j-1] yang sama dengan *suffix pattern*  dari P[1..j-1] dan sejajar dengan *text* bersangkutan (bisa dipermudah dengan table Border Function)
   3. Jika terjadi *mismatch* di indeks j =0, maka indeks i digeser 1 dan indeks j dimulai dari 0.

Kompleksitas Total O(m+n).

**Algoritma Boyer-Moore**

1. Definisikan table Last Occurrence mengenai informasi kemunculan terakhir suatu karakter pada P. Untuk P yang tidak ada di P, definisikan dengan -1.
2. Algoritm BM akan melakukan pencarian dari indeks Pattern dari kanan ke kiri P[len(pattern)..0]
3. Selama belom ditemukan atau indeks *teks(*i) belom lebih dari panjang *teks*(len(i)), maka
   1. Jika teks T[i] tidak ada di table Last Occurrence, maka definisikan dengan -1
   2. Jika teks T[i] nilainya sama dengan Pattern P[j], maka
      1. Jika indeks Pattern j sudah 0, pemeriksaan berhenti. Pattern sudah matching
      2. Jika belom, indeks i dan j sama sama digeser sekali ke kiri
   3. Jika indeks teks lebih kecil nilainya dengan yang di table Last Occurrence, maka lakukan i new = i + m- (io+1)
   4. Jika indeks teks lebih besar nilainya dengan yang di table Last Occurrence, maka lakukan i new = i + m- j

Kompleksitas Total O(mn+A).

**Regex**

Pada tugas ini, kita mendefinisika regex yang sudah disediakan oleh Bahasa Python. Adapun konstruksi kelas dilakukan dengan beberapa pendekatan. Untuk konstruksi pembentukan kelas karakter, maka kita melakukan ketentuan ini.

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

Untuk mencari pola pada posisi tertentu di regex yang kita definisikan maka kita menggunakan Boundary Matchers. Adapun Boundary Matchers dari regex adalah sebagi berikut.

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

Untuk mendefinisikan pengulangan maka kita menggunakan Quantifier sebagai berikut

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

**KODE**

**Algoritma KMP**

Pertama kita definisikan terlebih dahulu function untuk menghitung Prefix dan Suffix suatu keyword.

|  |
| --- |
| def tabelPrefix(keyword):  tabel = []  i=1  while i<len(keyword):  tabel.append(keyword[0:i])  i = i+1  return tabel  def tabelSufix (keyword):  tabel = []  i=1  while i<len(keyword):  tabel.append(keyword[len(keyword)-i:len(keyword)])  i = i+1  return tabel |

Kemudian kita bentuk Border Function dari keyword masukkan .

|  |
| --- |
| def tabelBorder(keyword):  tabel = []  idx=1  while idx<=len(keyword):  tabel\_prefix = tabelPrefix(keyword[0:idx])  tabel\_sufix = tabelSufix(keyword[0:idx])  i=len(tabel\_prefix)-1  while i>=0:  if (tabel\_prefix[i]==tabel\_sufix[i]):  break  i = i-1  tabel.append(i+1)  idx= idx+1  tabel.pop()  tabel.insert(0,0)  return tabel |

Setelah itu barulah kita definisikan Algoritma KMP.

|  |
| --- |
| def KMP(keyword, text):  tabel\_border = tabelBorder(keyword)  idx\_text = 0  idx\_keyword = 0  found = 0  while idx\_text<len(text):  if (keyword[idx\_keyword] == text[idx\_text]) :  idx\_keyword+=1  idx\_text += 1  elif idx\_keyword!=len(keyword) and idx\_keyword!=0:  idx\_keyword = tabel\_border[idx\_keyword]  else:  idx\_text +=1  idx\_keyword = 0  if idx\_keyword==len(keyword):  found = 1  break  return found  def searchKeywordKMP(stringInput, text):  i=0;  idx = []  for sentence in text:  hasil = KMP(stringInput,sentence)  if (hasil==1) :  idx.append(i)  i=i+1  return idx |

**Algoritma Boyer-Moore**

Pertama kita definisikan terlebih dahulu function untuk menghitung tabel Last Occurrence dari Algoritma Boyer-Moore.

|  |
| --- |
| def tabelLo(keyword): #kalau  tabel = {}  i= 0;  while i<len(keyword):  tabel[keyword[i:i+1]] = i  i = i+1  return tabel |

Selanjutnya kita bentuk Algoritma Boyer-Moore.

|  |
| --- |
| def BoyerMoore(keyword, text):  found =0  tabel\_lo = tabelLo(keyword)  # print(tabel\_lo)  idx\_text = len(keyword)-1  idx\_keyword = len(keyword)-1  while idx\_text < len(text) and idx\_keyword>=0:  if text[idx\_text:idx\_text+1] not in tabel\_lo:  idx\_text +=len(keyword)  idx\_keyword = len(keyword)-1  else:  if (keyword[idx\_keyword:idx\_keyword+1]==text[idx\_text:idx\_text+1]):  if (idx\_keyword>0):  idx\_keyword -=1  idx\_text -=1  else:  found =1  break  else:  idx\_text = idx\_text+len(keyword) - min((tabel\_lo[text[idx\_text:idx\_text+1]]+1),idx\_keyword)  idx\_keyword = len(keyword)-1    return found def searchKeywordBoyerMoore(stringInput, text):  i=0;  idx = []  for sentence in text:  hasil = BoyerMoore(stringInput,sentence)  if (hasil==1) :  idx.append(i)  i=i+1  return idx |

**Regex**

Lakukan import re untuk mengimport regex bawaan Bahasa Python. Definisikan beberapa dictionary yang nantinya diperlukan untuk pembentukkan regex.

|  |
| --- |
| import re;  hari = '(senin|selasa|rabu|kamis|jumat|sabtu|minggu|sen|sel|rab|kam|jum|sab|ming|monday|teusday|wednesday|thursday|friday|saturday|sunday)'  bulan =  '(januari|februari|maret|april|mei|juni|juli|agustus|september|oktober|november|desember|jan|feb|mar|apr|jun|jul|agu|ags|sept|sep|okt|nov|des|january|february|march|april|may|june|july|august|september|october|november|december|jan|feb|mar|apr|may|jun|jul|aug|sep|oct|nov|dec)'  angka = '(satu|dua|tiga|empat|lima|enam|tujuh|delapan|sembilan|puluh|ratus|ribu|ratus|juta|ribu|belas|sebelas|sepuluh|one|two|three|four|five|six|seven|eight|nine|ten|eleven|twelve|thirteen|fourteen|fifteen|sixteen|seventeen|eighteen|nineteen|twenty|thirty|fourty|fifty|sixty|seventy|eighty|ninety|hundred|thousand|milion)'  ket = '(sebelumnya|setelahnya|berlalu|usai|yang lalu|yang akan datang|lalu)'  ket\_orang = '(orang|korban|korban jiwa|ODP|PDP|orang ODP|orang PDP|kasus|people|person|pasien)'  ket\_number = '(st|rd|nd|th)' |

Fungsi untuk menghitung tanggal dengan regex

|  |
| --- |
| def searchDate(stringInput):  patterns = []  *#Pattern untuk “2 hari yang lalu”*  patterns.append('((('+ angka + '\s)\*)|(se)\*)hari(\s' + ket +'\*)')  *#Sabtu (11-1-2000) | Sabtu, 11/1/2000*  patterns.append(hari+'\s\(\*(([3][0-1]|[0-2]\*[0-9])(\/|\-)([0]\*[1-9]|[1][0-2])(\/|\-)\*(\d{4})\*)\)\*')  *#Sabtu, 11-1-2000 | Sabtu, 11/1/2000*  patterns.append(hari+'\,\s(([3][0-1]|[0-2]\*[0-9])(\/|\-)([0]\*[1-9]|[1][0-2])(\/|\-)(\d{4})\*)')  *#11-1-2000*  patterns.append('(([3][0-1]|[0-2]\*[0-9])(\/|\-)([0]\*[1-9]|[1][0-2])(\/|\-)(\d{4}))') #11-1-2000  *#Sabtu (11 April 2000)*  patterns.append(hari+'\s\(\*(([3][0-1]|[0-2]\*[0-9])('+ket\_number+'\*)\s'+bulan+'\s(\d{4})\*)\)\*')  *#Sabtu, 11 April 2000*  patterns.append(hari+'\,\s(([3][0-1]|[0-2]\*[0-9])('+ket\_number+'\*)\s'+bulan+'\s(\d{4})\*)')  *#11 April 2000*  patterns.append('(([3][0-1]|[0-2]\*[0-9])('+ket\_number+'\*)\s'+bulan+'\s(\d{4})\*)')  *#13/1/14 (DD MM YY)| 12-18-2014 (MM DD YY)*  patterns.append('(\d{1,})(\/|\-)(\d{1,})(\/|\-)(\d{1,})')  for pattern in patterns:  hasil = re.search(pattern,stringInput)  if (hasil):  break  if (hasil):  return hasil[0]  else:  return ''; |

Fungsi untuk menghitung waktu dengan regex

|  |
| --- |
| def searchTime(stringInput):  patterns = []  # Pattern untuk pukul\* 5.06 WIB\* | pukul 5:03 WITA  patterns.append('((pukul\s)\*\w{1,}(:|.)\w{1,}\swi\w{1,})')  #05:03  patterns.append('(\w{1,}:\w{1,})(:\w{1,})\*')  for pattern in patterns:  hasil = re.search(pattern,stringInput)  if (hasil):  break  if (hasil):  return hasil[0]  else:  return ''; |

Fungsi untuk menghitung jumlah orang dalam regex dan mengambil jumlah yang paling dekat dengan keyword yang ada

|  |
| --- |
| def searchJumlah(keyword, stringInput):  searchkeyword = re.search(keyword,stringInput)  if searchkeyword!=None:  idxkeyi, idxkeyj = searchkeyword.span()  else:  return 0  min\_nilai = -999  min = 9999  patterns = []  patterns.append('([\d][\d]\*[\.|\,]\*[\d]\*[\.|\,]\*[\d]\*[\s]'+ket\_orang+')')  for pattern in patterns:  hasils = re.findall(pattern,stringInput)  for hasil in (hasils):  if hasil!=None and str(hasil)!=str(keyword):  i, j = re.search(str(hasil[0]),stringInput).span()  if abs(i-idxkeyi)<min:  min\_nilai = hasil[0]  min = abs(i-idxkeyi)  if (min\_nilai!=-999 ):  return min\_nilai  else:  return 0; |

Beberapa fungsi tambahan yaitu fungsi untuk memisahkan per kalimat dalam text dan fungsi untuk mencari keyword dengan metode regex

|  |
| --- |
| def splitTexttoSentence(text):  hasil = text.split('. ')  return hasil  def searchKeywordRegex(stringInput, text):  i=0;  idx = []  for sentence in text:  hasil = re.search(stringInput,sentence)  if (hasil) :  idx.append(i)  i=i+1  return idx |

Fungsi main Program

Kalkulasi akan memasukkan nilai yang tepat ke list. Selanjutnya akan diolah di mainProgram untuk dipanggil oleh flask web.

|  |
| --- |
| def kalkulasi(hasil, split, index, splitnormal, keyword, folder):  hasil.append("\n")  hasil.append(("--------- Diambil dari : "+ str(folder) + " --------- "))  if (searchDate(split[index])):  hasil.append(searchDate(split[index]))  else:  hasil.append(searchDate(split[0]))  if (searchTime(split[index])):  hasil.append(searchTime(split[index]))  else:  hasil.append(searchTime(split[0]))  hasil.append("Jumlah : "+ str(searchJumlah(keyword.lower(),split[index])))  hasil.append(splitnormal[index])  return hasil  def mainProgram(folder, keyword, jenis):  folder\_ = '../test/'+folder  file = open(folder\_, 'r')  stringnormal = file.read().replace('\n',' ')  splitnormal = splitTexttoSentence(stringnormal)  string = stringnormal.lower()  split = splitTexttoSentence(string)  hasil = []    if jenis=="Boyer-Moore":  for index in searchKeywordBoyerMoore(keyword.lower(),split):  hasil = kalkulasi(hasil, split, index, splitnormal, keyword, folder)  elif jenis=="KMP":  for index in searchKeywordKMP(keyword.lower(),split):  hasil = kalkulasi(hasil, split, index, splitnormal, keyword, folder)  else:  for index in searchKeywordRegex(keyword.lower(),split):  hasil = kalkulasi(hasil, split, index, splitnormal, keyword, folder)  return hasil |

Kode untuk Web.py berisi penggunaan kakas flask

|  |
| --- |
| #set FLASK\_APP=Web.py  #flask run  from flask import Flask,render\_template  from flask import request, redirect, url\_for  from Algoritma import mainProgram  app = Flask(\_\_name\_\_)  @app.route("/", methods = ["POST","GET"])  def home():  if request.method =="POST":  return redirect(url\_for('form'))  else:  return render\_template('home.html')  @app.route("/form", methods = ["POST", "GET"])  def form():  if request.method =='POST':  if request.form['submit']== 'oke':  foldr = request.form["folder"]  key =request.form["keyword"]  algo =request.form["algo"]  if foldr!='':  hasil = mainProgram(foldr,key,algo)  return render\_template('form.html', error= hasil)  else:  return render\_template('form.html')  elif request.form['submit']== 'home':  return redirect(url\_for('home'))  else:  return redirect(url\_for('about'))  else:  return render\_template('form.html')  @app.route("/about", methods = ["POST", "GET"])  def about():  if request.method =="POST":  if request.form['submit']== 'home':  return redirect(url\_for('home'))  else:  return render\_template('about.html')    if \_\_name\_\_ == '\_\_main\_\_':  app.run(debug = True) |

web.html

|  |
| --- |
| <html>  <head>  <!-- <link rel= "stylesheet" type= "text/css" href= "{{ url\_for('static',filename='styleshome.css') }}"> -->  <link rel= "stylesheet" type= "text/css" href= "static/styles/styleshome.css">  <title>Kevin Stefano</title>  </head>  <body>  <style>  body{  background-image: url("static/img/background.gif");  background-size: cover;    }  </style>  {% block content %}  {% endblock content %}  </body>  </html> |

form.html

|  |
| --- |
| {% extends "web.html" %}  {% block titile %}Form Page{% endblock %}  {% block content %}  <div class= "form">  <form action = "{{url\_for('form')}}" method="POST"> <br><br>  <style>  p{  font-size: large;  }  input {  font-size: large;  text-align: center;  }  select{  font-size: large;  text-align: center;  }  </style>  <p style="color:white" style="font-size:100px">Folder : (Contoh: berita1.txt)</p>  <p style="color:white"></p><input type="text" id="folder" name="folder"><br><br></p>    <p style="color:white"><label for="keyword">Keyword:</label></p>  <input type="text" id="keyword" name="keyword"><br><br>    <p style="color:white"><label for="algoritma">Choose Algoritma:</label></p>  <select id="algo" name="algo">  <option label="Regex">Regex</option>  <option label="Boyer-Moore">Boyer-Moore</option>  <option label="KMP">KMP</option>  </select>  <input type="submit" name="submit" value="oke"><br><br><br>    {% if error %}  {% for hasil in error %}  <p class=error style="color:white" > {{ hasil }} </p>  {% endfor %}  {% endif %}    <input type="submit" name="submit" value="home">  <input type="submit" name="submit" value="about"><br>  </form>  </div>    {% endblock content %} |

home.html

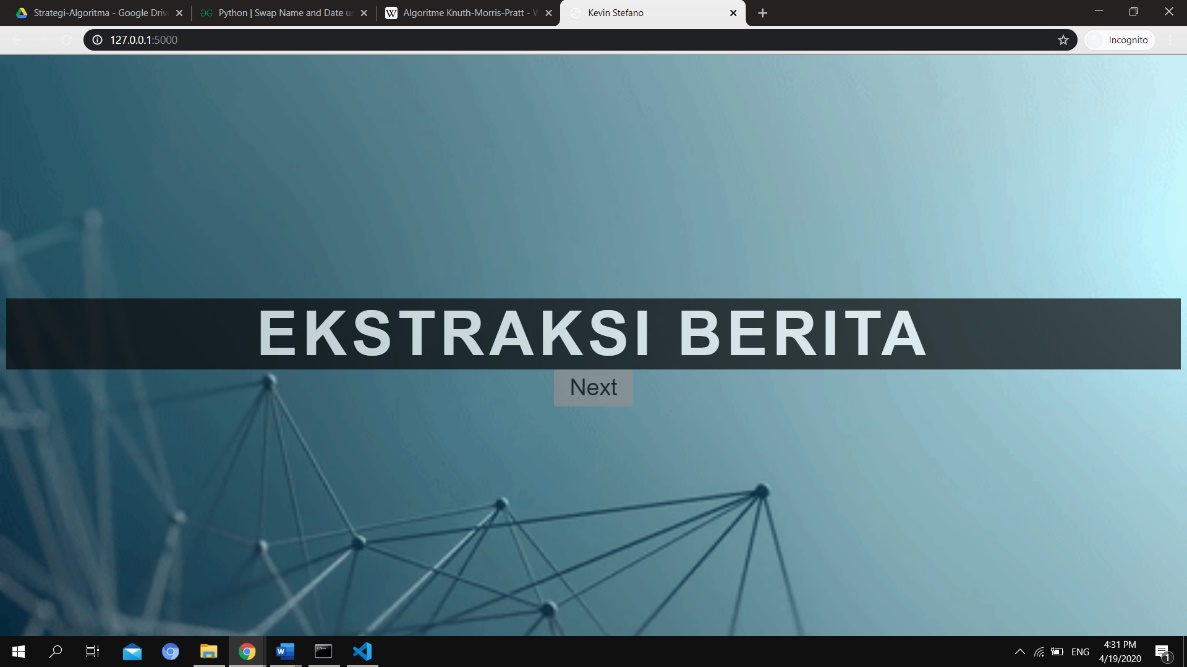
|  |
| --- |
| {% extends "web.html" %}  {% block titile %}Form Page{% endblock %}  {% block content %}  <div class="container">  <form action = "#", method="POST">  <a class="text">EKSTRAKSI BERITA</a>  <button class="button" value="">Next</button><br>  </div>  {% endblock content %} |

about.html

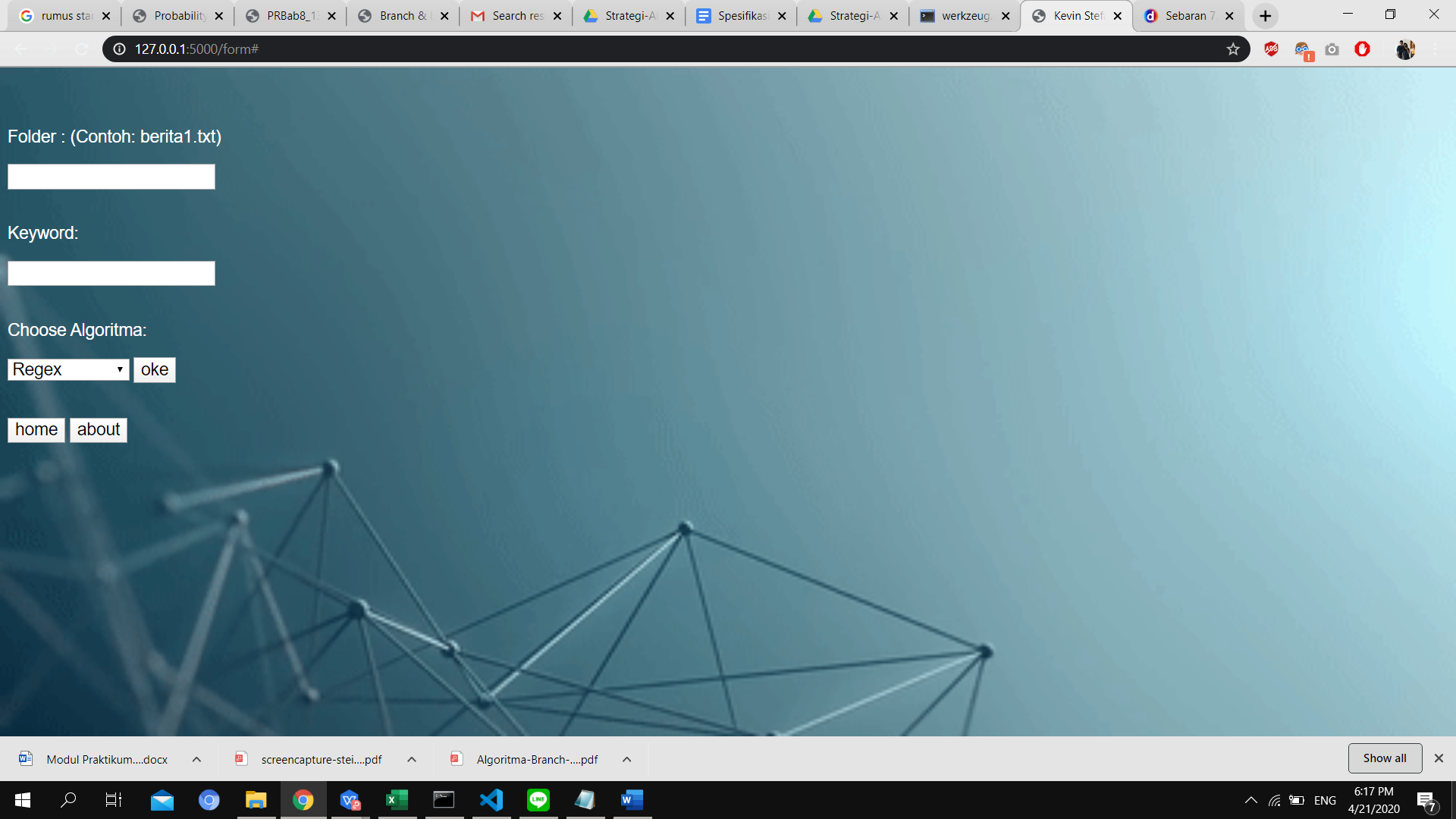
|  |
| --- |
| {% extends "web.html" %}  {% block titile %}Form Page{% endblock %}  {% block content %}  <style>  h1{  color: white;  text-align: center;  }  p{  font-size: large;  text-align: center;  }  input{  font-size: large;  text-align: center;  }  </style> <br> <br><br><br><br><br><br>  <h1>CREATE BY :</h1>  <p style="color:white" style="font-size:100px" id="name">Kevin Austin Stefano<br>13518104</p>  <br> <br><br><br><br><br><br>  <form action = # method="POST">  <input type="submit" name="submit" value="home" style="margin:0px auto; display:block;"><br>  </form>  {% endblock content %} |

**CONTOH IMPUT-OUTPUT**

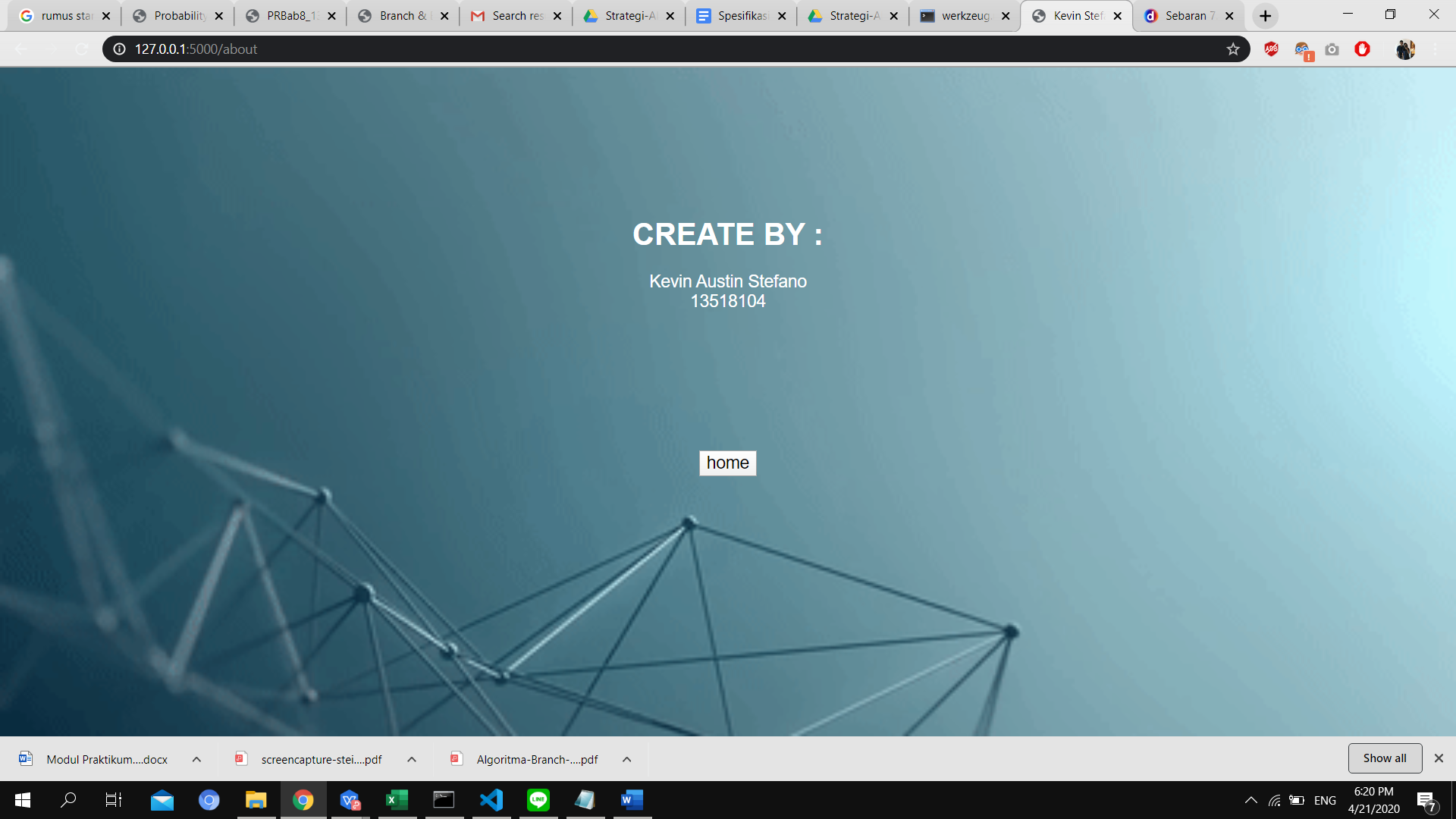
Tampilan awal Web / home



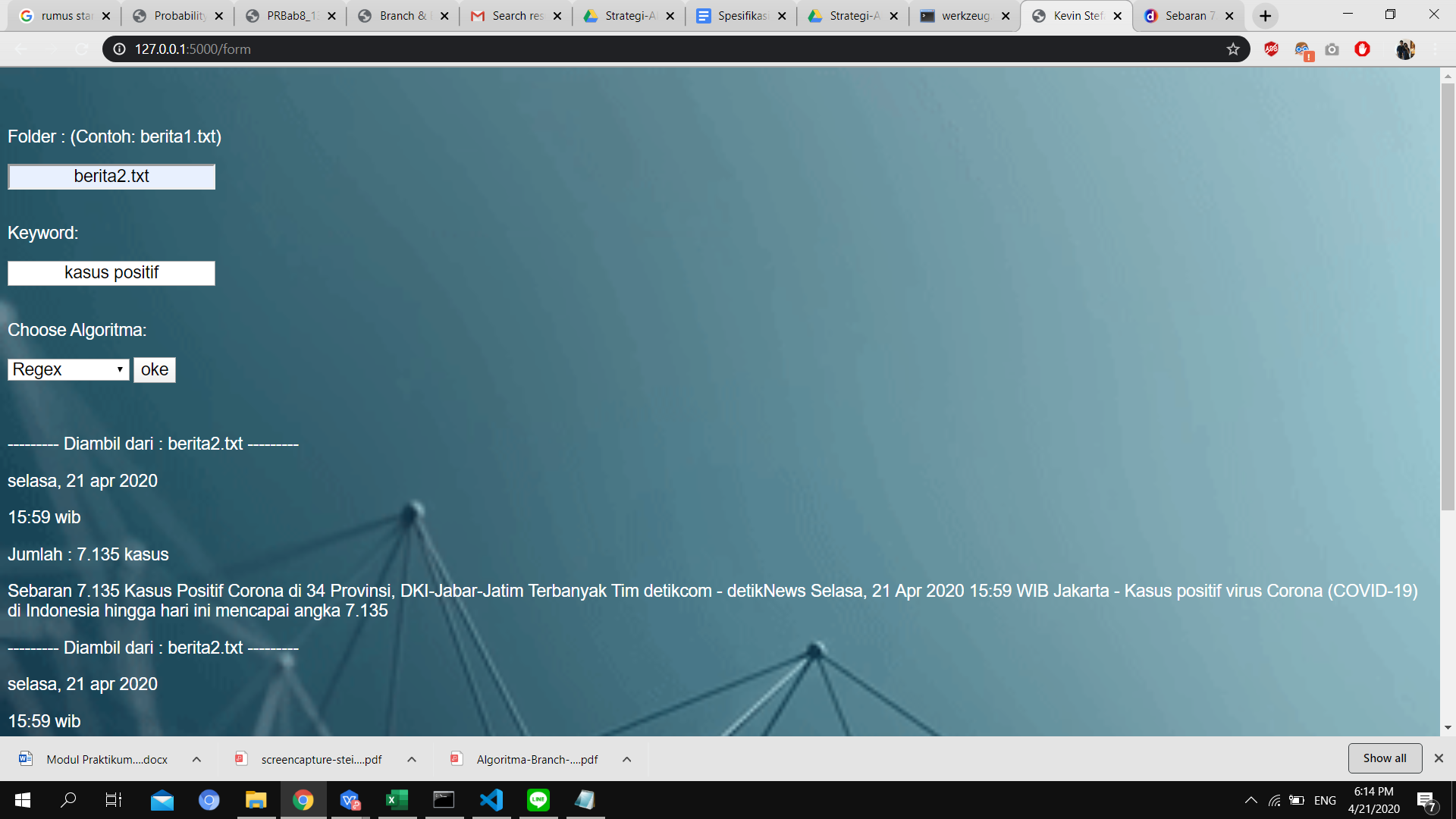
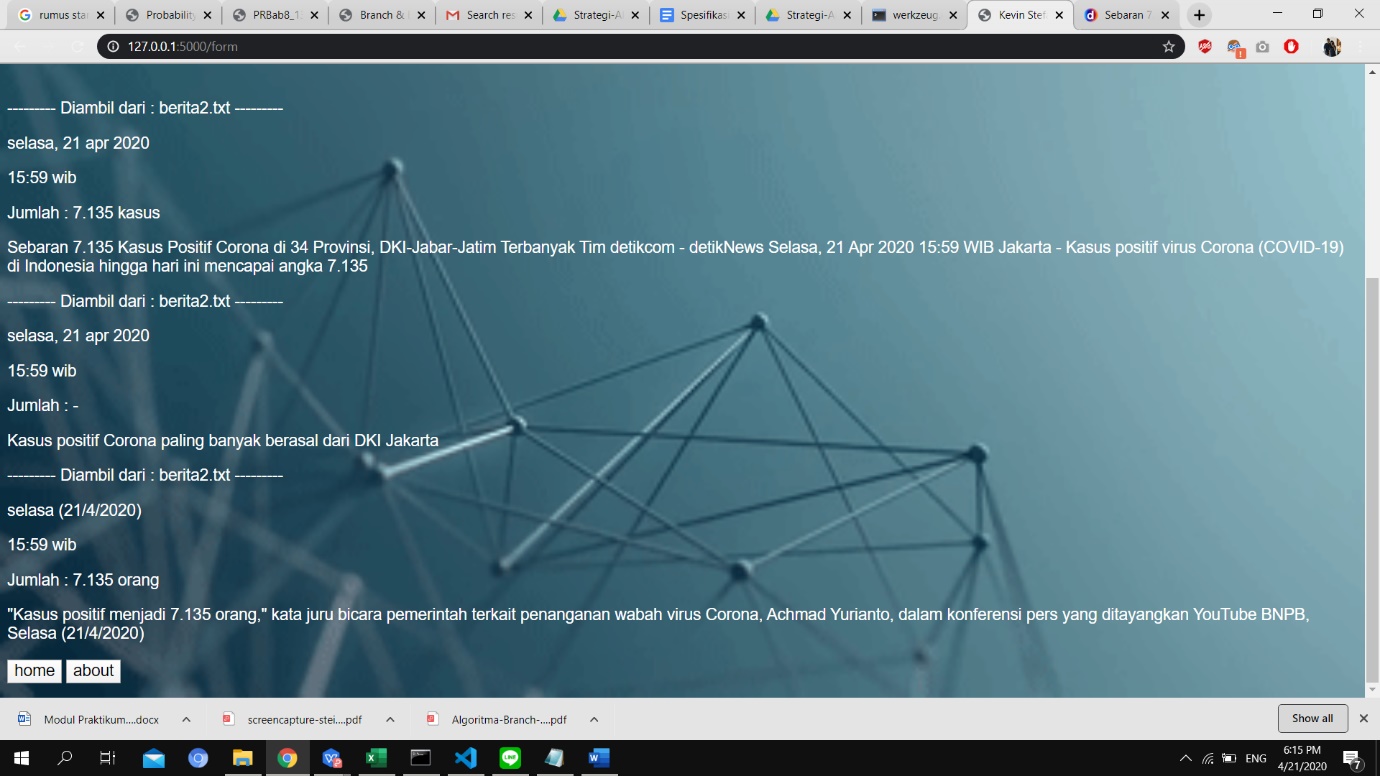
Contoh tampilan form



Contoh tampilan about

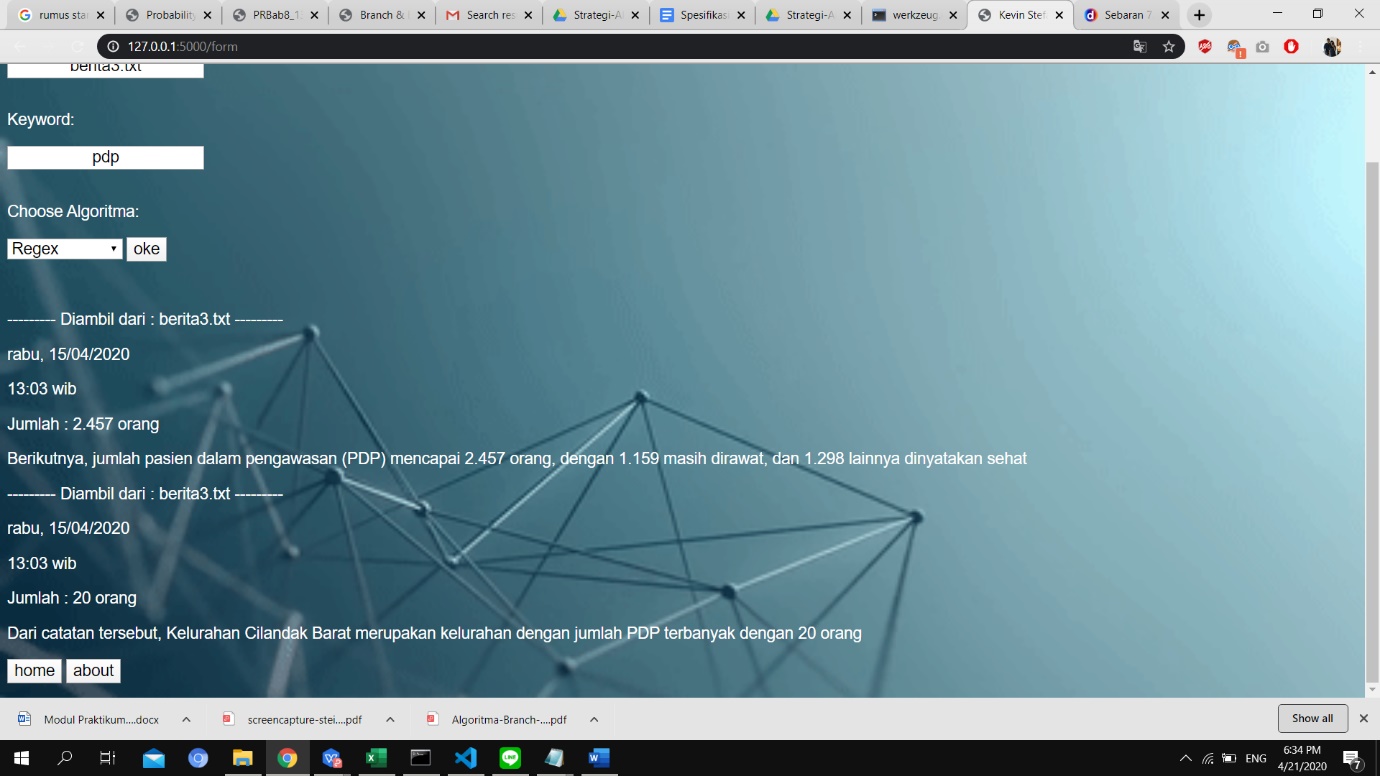


Contoh input

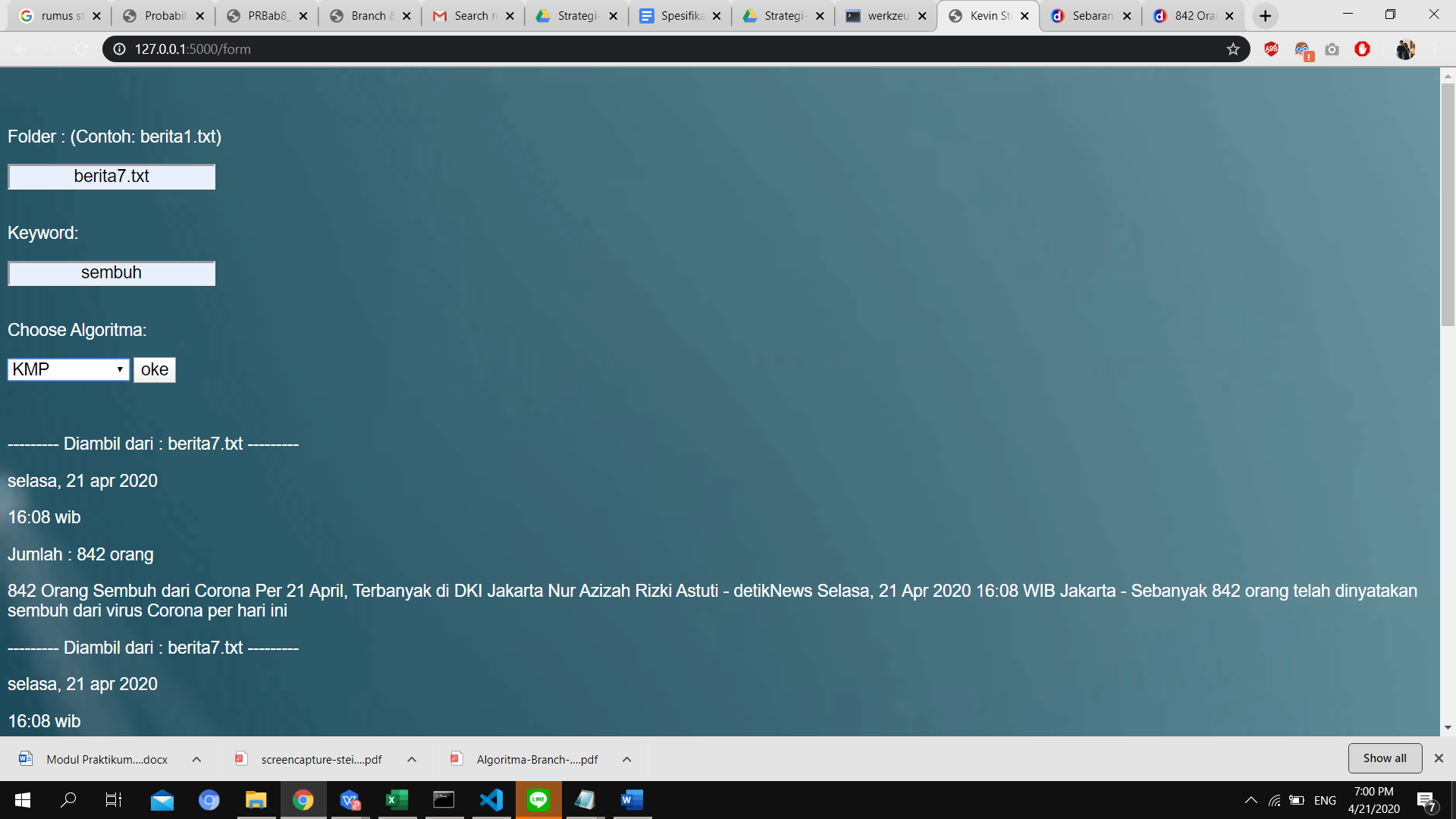


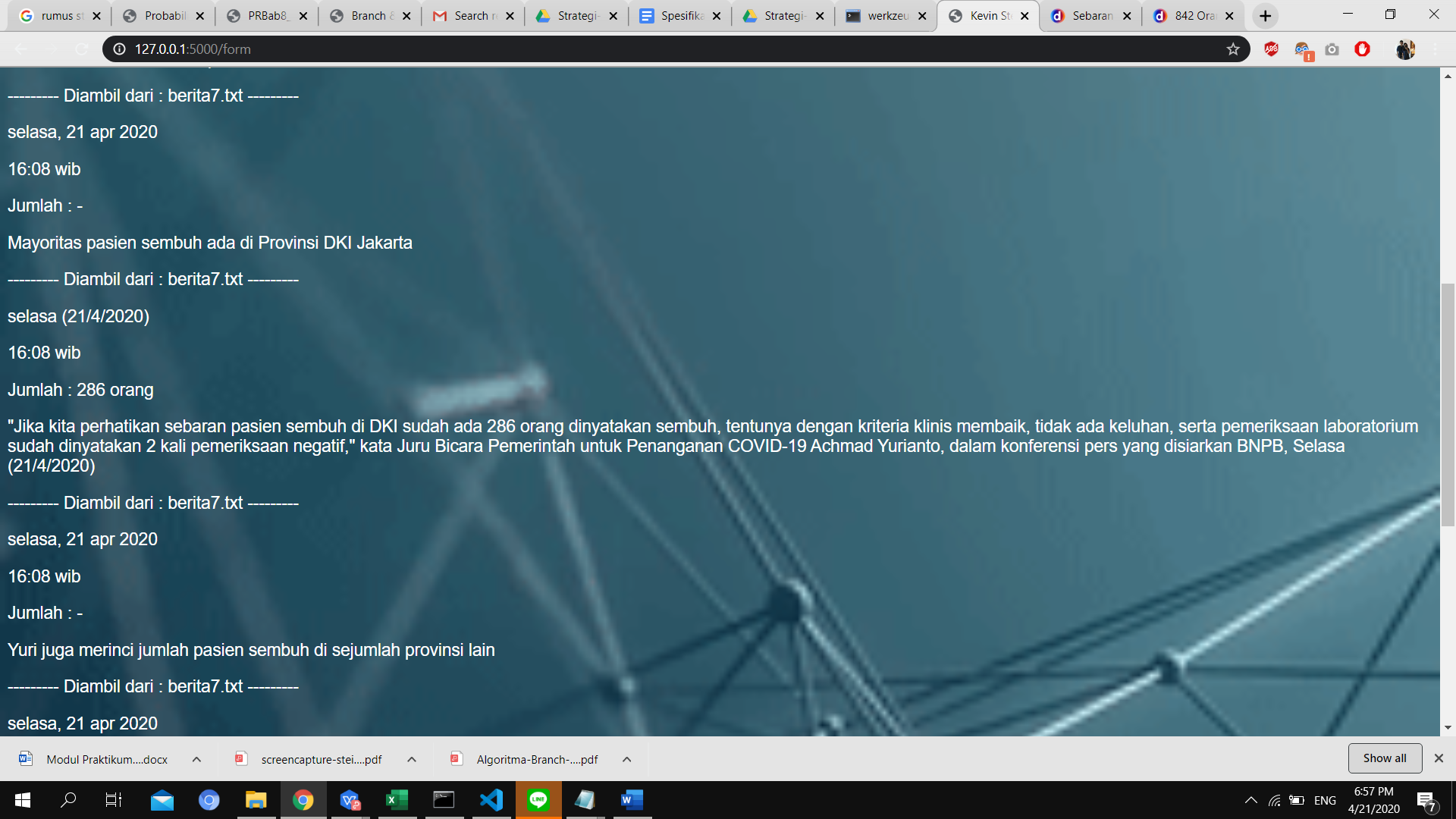
Contoh input 2

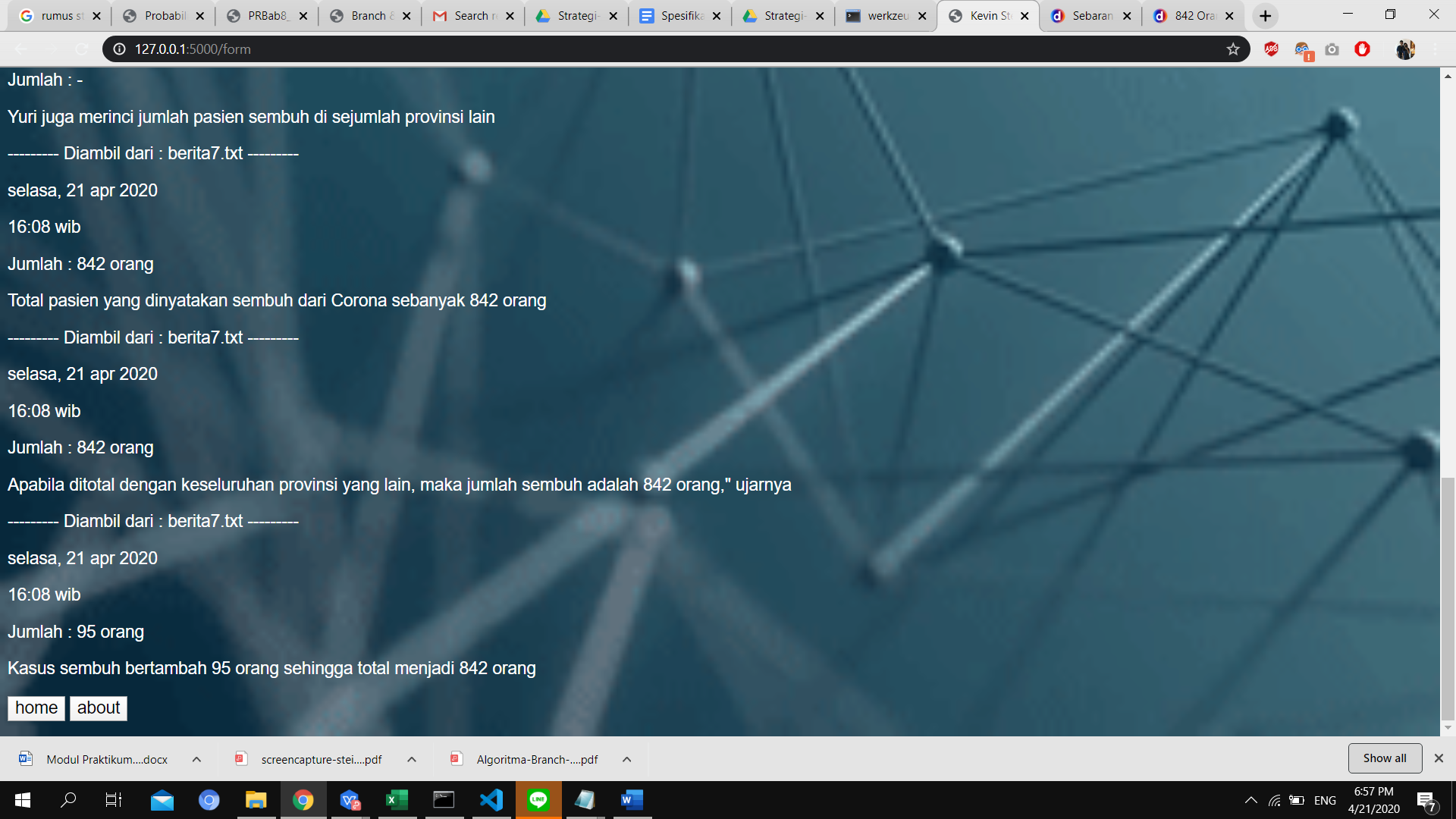




Contoh input 3:







**KESIMPULAN**

Berdasarkan algoritma yang dipakai algoritma yang paling efisien adalah Algoritma Boyer-Moore. Teknik dan ide algoritma yang mana melakukan pembacaan dari akhir (*looking* glass) dan teknik lompatan (*character jump*) menyebabkan algoritma ini tergolong lebih efisien dibandingkan dengan algoritma lain seperti KMP maupun Regex yang melakukan pencocokkan string dan pembacaan dari kiri ke kanan. Namun Algoritma Boyer-Moore tetap belomlah algoritma yang paling efektif dalam *string matching,* masih ada algoritma lainnya yang lebih *advance* seperti Algoritma Wu-Manber dan lainnya.

|  |  |  |
| --- | --- | --- |
| 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 n | v |  |

**REFERENSI**

[1] Munir, Rinaldi. 2007. Diktat Kuliah IF2251 Strategi Algoritmik. Program Studi Teknik

Informatika STEI ITB.

[2] Knuth, Donald E. Morris, James H. Pratt, Vaughan R. 1977. Fast Pattern Matching in Strings, SIAM Journal of Computing Vol 6 No.2.

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

[4] Thompson, Ken, “Regular Expression Search Algorithm”, Communication of the ACM, Volume 11, Number 6, 1968, hal 419-422