Pemanfaatan Algorima Pencocokan *String* Dalam Pembuatan *Web Scraper* Sederhana

Juniardi Akbar, 13517075
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
juniardiakbar@gmail.com, 13517075@std.stei.itb.ac.id

***Abstrak*—*Web scraping* adalah proses mendapatkan data dengan cara mengekstrak struktur websitenya dengan menggunakan kode program. Untuk melakukan *web scraping*, diperlukan suatu alat yang disebut *web scraper* untuk mengelola data yang terdapat di suatu laman. Proses pengelolaan data dapat dilakukan dengan memanfaatkan algoritma pencocokan *string* untuk mendapatkan konten dari suatu tag HTML. Proses pencocokan string dapat dilakukan dengan menggunakan algortima Knut-Morris-Pratt, Boyer-Moore, dan *Regular Expression*.**

Kata kunci*— *web scraping*, *web scraper*, Knuth-Morris-Pratt, Boyer-Moore, *Regular Expression

I. PENDAHULUAN

Web scraping adalah proses mendapatkan data dengan cara mengekstrak struktur websitenya dengan menggunakan kode program. Secara mudahnya, web scraping adalah metode yang bisa digunakan untuk mendapatkan data-data yang ada di suatu laman internet. Misalnya, web scraping digunakan untuk mendapatkan data perkiraan cuaca dari laman BMKG, mendapatkan data waktu shalat dari laman Kemenag, dan lain-lain.[5]

Jika kita ingin mendapatkan data dari suatu laman namun laman tersebut tidak menyediakan *Application Programming Interface* (API) yang dapat memfasilitasi pertukaran informasi antar perangkat lunak, maka *web scraping* dapat dilakukan untuk mendapatkan data-data yang diinginkan. Secara umum, *web scraping* memiliki kegunaan untuk mempercepat proses pengumpulan data, membantu proses analisa data, dan menganalisa data kompetitor [1].

Terdapat beberapa teknik dalam melakukan web scraping, salah satunya adalah teknik *parsing HTML*. *Parsing HTML* adalah salah satu teknik yang paling umum digunakan. Pada metode ini, data-data yang tedapat dalam kode program HTML akan diolah untuk diambil data yang diinginkan. Proses ini cenderung lebih mudah karena kita hanya perlu mengidentifikasi data yang ingin diambil dengan cara memahami struktur HTMLnya saja.

Untuk melakukan *web scraping*, diperlukan sebuah alat yang disebut *web scraper*. Saat ini, terdapat beberapa *web scraper* yang dapat digunakan seperti *BeautifulSoup* yang menggunakan bahasa pemrogram pyhon. Pada makalah ini akan dibahas bagaimana pemanfaatan algoritma pencocokan *string* dalam pembuatan *web scraper* sederhana yang mampu menjalankan fungsionalitas-fungsionalitas sederhana seperti mendapatkan elemen berdasarkan id dan kelas.

II. LANDASAN TEORI

A. Algoritma Pencocokan String

Jika teks adalah *string* yang memiliki panjang n karakter dan *pattern* adalah *string* yang memiliki panjang m karakter dengan m yang jauh lebih kecil dibandingkan n, maka pencocokan string adalah proses mencari lokasi suatu patern dalam suatu teks. Aplikasi dari algoritma ini sangatlah luas dan beragam. Mulai dari pencarian di dalam *Text/Code Editor*, analisi citra, bahkan sampai ke pencocokan rantai asam amino pada rantai DNA dalam cabang keilmuan *bioinformatics*.

Gambar 1. Contoh Penerapan Algoritma Pencocokan String Pada Pencarian Rantai DNA
Sumber: Septu Jamasoka, IF2009

Algortima pencocokan string dibagi menjadi dua berdasarkan fungsionalitasnya. Pertama adalah algortima *exact matching* yaitu algoritma yang akan mencari keberadaan suatu patern yang benar-benar spesifik dalam teks. Algoritma yang akan digunakan dalam *exact matching* ini adalah Brute Force, Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM). Kedua adalah algortima *Regular Expression* yang mampu mencari keberadaan suatu patern walaupun tidak spesifik dalam text.

1. Algoritma *Brute Force* dalam *Exact Matching*

Algoritma *Brute Force* adalah algoritma yang lempang untuk memecahkan suatu permasalahan [4]. Algoritma *Brute Force* dalam *exact matching* memiliki langkah sebagai berikut.

- Awalnya *patern* terletak pada awal teks.
- Lakukan pencocokan setiap hurufnya dari kiri ke kanan.
- Jika terdapat ketidakcocokan, geser *patern* satu huruf ke kanan. Lakukan kembali langkah sebelumnya.
- Jika semua huruf cocok, maka ditemukan lokasi *patern* dalam teks dan algortima selesai.
- Jika patern sudah berada di akhir teks dan masih terapat ketidakcocokan, maka proses pencarian berakhir dan *patern* tidak ditemukan dalam teks.

Contoh penggunaan algoritma *Brute Force* pada teks "NOBODY NOTICED HIM" dengan *patern* "BODY" adalah sebagai berikut.

```
  NOBODY NOTICED HIM
1 BODY
2  BODY
3   BODY
```

Kompleksitas algoritma *Brutte Force* dalam *exact matching* ini dapat dihitung dari jumlah perbandingan yang dilakukn dengan *worst case* adalah O(mn) dan *best case* adalah O(n).

2. Algoritma KMP dalam *Exact Matching*

Algoritma Knuth-Morris-Pratt (KMP) adalah algoitma pencarian *pattern* dalam teks dengan pencarian dari kiri ke kanan seperti *brute force* namun dengan optimasi pada penggeseran posisi patern yang lebih pintar dibandingkan *brute force*. Algoritma ini dikembanagkaan oleh tiga orang yang berbada yaitu D. E. Knuth, J.H. Morris, dan V. R. Pratt. [4]

Algoritma KMP akan memproses ulang *patern* untuk menemukan kecocokan prefix antara *patern* dengan teks. Untuk menerapkan hal tersebut, diperlukan suatu fungsi pinggiran yang akan menentukan banyakanya pergeseran yang harus dilakukan ketika terjadi ketidakcocokan dalam perbandingn setiap hurufnya. Fungsi pinggiran pada KMP b(k) didefinisikan sebagai ukuran terpanjang dari prefix *patern* P[0..k] yang juga merupakan sufix dari P[1..k] dengan k adalah posisi sebelum ketiakcocokan (k = j-1) dengan j adalah posisi ketidakcocokan.

Contoh dari fungsi pinggiran pada algoritma KMP dalam *exact matching* pada *patern* "ABAABA" adalah sebagai berikut.

j	0	1	2	3	4	5
P[j]	A	B	A	A	B	A
k	-	0	1	2	3	4
b(k)	-	0	0	1	1	2

Tabel 1 Penerapan fungsi pinggiran pada algoritma KMP

Langkah-langkah algoritma KMP dalam *exact matching* adalah sebagai berikut.

- Awalnya *patern* terletak pada awal teks.
- Lakukan pencocokan setiap hurufnya dari kiri ke kanan.
- Jika terdapat ketidakcocokan, gunakan fungsi pinggiran pada pola. Lakukan pergeseran pada *patern* sehingga teks sekarang sejajar dengan P[b(k)] dan lakukan langkah sebelumnya.
- Jika semua huruf cocok, maka ditemukan lokasi patern dalam teks dan algortima selesai.
- Jika patern sudah berada di akhir teks dan masih terapat ketidakcocokan, maka proses pencarian berakhir dan patern tidak ditemukan dalam teks.

Contoh penggunaan algoritma KMP pada teks "ABACAABACCABACABAA" dengan *patern* "ABACAB" adalah sebagai berikut.

```
   ABACAABACCABACABAA
1 ABACAB
2     ABACAB
3      ABACAB
4          ABACAB
5           ABACAB
```

Kompleksitas algoritma KMP dalam *exact matching* ini dapat dihitung dari penjumlahan kompleksitas menghitung fungsi pinggiran dan kompleksitas pencocokan *string*. Untuk kompleksitas menghitung fungsi pinggiran dari pola dengan m karakter adalah O(m) dan kompleksitas pencarian string pada teks dengan n karakter adalah O(n). Sehingga, kompleksitas algoritma KMP adalah O(m+n)

3. Algoritma BM dalam *Exact Matching*

Algortima Boyer-Moore (BM) adalah algoritma pencarian *pattern* dalam teks dengan pencarian dari kanan ke kiri namun dengan pengecekaan pola yang tetap dari kiri dengan optimasi pada pergeseran posisi patern yang akan melompat sejauh mungkin. [4]

Pada tahun 1979, Zvi Galil berhasil mengoptimasi algoritma ini sehingga dapat menghasilkan runtime linier untuk setiap kasus. Algoritma BM memproses *patern* dengan membandingkan setiap karakter pada *patern* namun melewatkan bagian yang diketahui cocok.

Sama seperti algoritma KMP, algoritma BM ini juga melakukan pemrosesan awal terhadap *patern* P dengan membuat fungsi kemunculan terakhir L() yang didefiniskan sebagai indeksk terbesar i sehingga P[i] = x untuk x di dalam A atau -1 jika untuk semua i, berlaku P[i] ≠ x.

Contoh dari fungsi kemunculan terakhir pada algoritma BM dalam *exact matching* pada *patern* "ABACAB" dan A = {a,b,c,d} adalah sebagai berikut.

x	a	b	c	d
L(x)	4	5	3	-1

Tabel 2 Penerapan fungsi kemunculan terakhir pada algoritma BM

Langkah-langkah algoritma BM dalam *exact matching* adalah sebagai berikut.

- Awalnya *patern* terletak pada awal teks.
- Lakukan pencocokan setiap hurufnya dari kanan ke kiri.
- Jika terdapat ketidakcocokan, gunakan fungsi kemunculan terakhir. Lakukan pergeseran pada *patern* sebanyak mungkin sesuai dengan L(x).
- Jika semua huruf cocok, maka ditemukan lokasi patern

dalam teks dan algortima selesai.

- Jika patern sudah berada di akhir teks dan masih terapat ketidakcocokan, maka proses pencarian berakhir dan patern tidak ditemukan dalam teks.

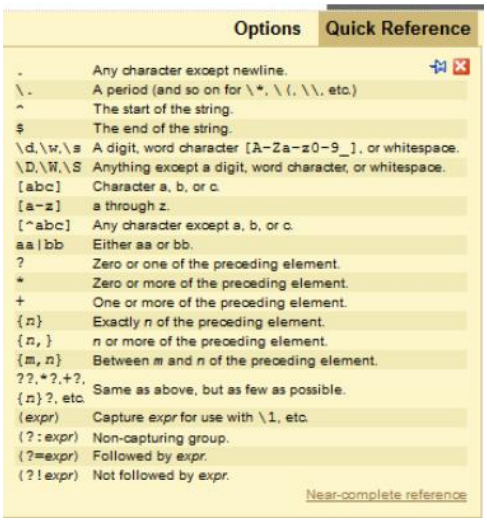
Contoh penggunaan algoritma BM pada teks "ABACAABADCABACABAA" dengan *patern* "ABACAB" adalah sebagai berikut.

```
   ABACAABADCABACABAA
1  ABACAB
2   ABACAB
3    ABACAB
4     ABACAB
5           ABACAB
6            ABACAB
```

Kompleksitas algoritma BM dalam *exact matching* ini dapat dihitung dari jumlah perbandingan yang dilakukan dengan *worst case* adalah O(nm+A) dan *best case* adalah O(m/n).

4. Algoritma *Regular Expression*

Algoritma *Regular Expression (regex)* adalah algoritma yang mendefenisikan *patern* sebagai rantai karakter. Pada *regex* pencarian yang dihasilkan bukanah *exact matching* sehingga *patern* tidak harus sama pesis terhadap suatu teks. Pola-pola dalam *regex* secara umum didefinisikan dengan sintaks yang sebagai berikut.



Gambar 2. Sintaks Regular Expression
Sumber: http://regexpal.com.s3-website-us-east-1.amazonaws.com/?_ga=2.259623882.1129009230.1526062487-623396008.1526062487

Sehingga, dengan menggunakan sintaks di atas, jika kita memasukan *patern* "`/.udi/g`" maka itu sama saja dengan mencari semua kata yang mengandung kata "udi" dengan tambahan minimal satu huruf di depannya misalnya budi, rudi, atau yudi.

Selain itu regex juga dapat digunakan untuk mencari sesuatu yang sulit seperti contohnya "`(\(?\d{3}\)?[ -])+\d{4}`" maka itu sama saja dengan mencari semua teks yang sama dengan semua nomor telepon. Selain itu, jika kita memasukkan *pattern* "`[\w-]+ ?(@|\(?at\)?) ?(([\w-]+(\.| ?dot ?))+[a-zA-Z]{2,4}`" maka itu sama saja dengan mencari semua alamat email.

Pada beberapa bahasa pemrograman, regex dapat digunakan dengan mengimport library yang tersedia. Contoh bahasa pemrograman yang dapat melakukan regex adalah Java dan Python. Khusu pada python, regex dijalankan dengan mengimport library "re".

B. *HyperText Markup Language (HTML)*

Secara mudah, HTML adalah kode yang digunakan untuk mengatur tata letak tampilan halaman web dan isinya. Secara teknis, HTML bukanlah sebuah bahasa pemrograman melainkan sebuah *markup language* yang mendefenisan suatu konten dalam serangkaian suatu tag yang digunakan untuk membungkus konten tersebut [2]. Konten akan dibungkus dalam sebuah tag yang dimulai dengan tanda <nama tag> dan diakhiri dengan tag </nama tag>. Isi dari konten yang disertai dengan tag disebut dengan elemen.

Berikut ini adalah contoh dari sebuah sintaks HTML yang sangat sederhana.

```
<!DCODE html>
<html>
<head>
<title> Judul halaman </title>
</head>
<body>
  <h1> Judul Laman </h1>
  <p> Isi laman </p>
</body>
</html>
```

Pada contoh di atas, terdapat 6 buah tag dengan kegunaan yaitu sebagai berikut.

- Tag html berfungsi menandakan elemen root.
- Tag head berfungsi menyimpan informasi awal dari suatu laman.
- Tag title berfungsi menyimpan judul dari suatu laman.
- Tag body berfungs menyimpan konten utama (isi) dari suatu laman.
- Tag h1 berfungsi menyimpan judul dari suatu konten utama.
- Tag p berfungsi menyimpan konten suatu laman dalam bentuk paragraph.

Selain enam buah tag di atas, masih banyak sekali tag yang terdapat di HTML. Setiap tag memiliki fungionaltas dan menyimpan tata letak dan tampilan yang berbeda pula. Contohnya adalah tag <img> yang berguna untuk menampilkan gambar, tag <li> untuk menampilkan list, dan lain-lain.

HTML memiliki atribut yang berfungsi untuk menyimpan informasi ekstra dari suatu elemen. Selain itu, atribut biasa juga digunakan untuk mengidentifikasi dan membedakan suatu elemen yang memiliki nama tag yang sama. Sebuah atribut harus mempunyai nama atribut diikuti dengan sama dengan, petik buka, nilai atribut, lalu diikuti dengan petik tutup. Untuk kali ini, pembahasan yang akan dilakukan hanya berkisar pada penggunaan atribut class dan id.

Atribut class dan adalah atribut yang memiliki fungsi utama untuk mengidentifikasi suatu elemen HTML. Perbedaan antara class dan id paling terlihat pada jumlah nilai atribut yang dimiliki oleh elemen lainnya. Pada atribut id, tidak boleh terdapat elemen HTML yang memiliki nilai atribut yang sama. Sedangkan pada atribut class, nilai atribut boleh dimiliki oleh lebih dari satu atribut.

Untuk mendapatkan suatu elemen dari sebuah struktur penuh HTML, kita dapat membagi-bagi kumpulan elemen tadi sehingga hanya akan tersisa elemen yang akan diolah untuk diambil konten yang diinginkan saja. Proses ini cenderung lebih mudah karena kita hanya perlu mengidentifikasi data yang ingin diambil dengan cara memahami struktur HTMLnya saja.

Misalkan terdapatkan contoh sintaks HTML yang sederhana seperti berikut.

```
...
<body>
  <h1> Judul Laman </h1>
  <div class=”konten”>
    <p> Isi laman </p>
    <div class=”share”>
      <p> Postingan ini telah dishare sebanyak
</p>
      <p id=”bykshare”> 200 kali </p>
    </div>
  </div>
</body>
...
```

Pada contoh di atas, jika kita hanya ingin mendapatkan data jumlah share, kita dapat mengabaikan semua data pada html di atas dan hanya berfokus pada elemen yang memiliki id=”bykshare” yang terletak pada elemen `<p id=”bykshare”> 200 kali </p>`.

C. *Web Scraper*

Beberapa teknik pemecahan konten yang umumnya dipakai dalam dalam melakukan *web scraping* adalah sebagai berikut. [3]

- HTML *Parsing*, yaitu teknik yang paling umum dipakai dalam web scraping. Pada proses ini format HTML diubah sesuai dengan yang diinginkan sehingga proses pencariannya dapat menggunakan query.
- *Patern Matrching*, yaitu proses menentukan pola yang ingin dicari dari suautu halaman html utuh. Teknik ini akan mencari keberadaan suatu elemen dengan atribut yang sama dengan *patern* yang dimasukkan.
- DOM *Parsing*, yaitu teknik yang digunakan untuk mengekstrak seluruh halaman web atau hanya sebagian saja, bahkan jika konten yang dihasilkan bersifat dinamis. Teknik ini dilakukan dengan memanfaatkan browser yang bisa mengambil data yang umumnya dalam format *Extensible Markup Language* (XML).

Pada saat ini sudah banyak cara yang dapat kita gunakan untuk membuat web scraper. Bahkan sudah banyak sekali web scraper yang tersedia seperti yang paling terkenal adalah *BeautfulSoup*. Pada kali ini, penulis akan mencoba untuk memanfaatkan algoritma-algoritma pencocokan string untuk melakukan web scraping secara sederhana.

III. PEMANFAATAN ALGORITMA PENCOCOKAN PATERN PADA PEMBUATAN WEB SCRAPER SEDERHANA

A. Pembuatan *Web Scraper*

Dengan menggunakan *web scraper* sederhana yang penulis buat, proses pencarian elemen html hanya dapat dilakukan dengan mencari berdasakan id dan class. Jika mencari berdasarkan id, maka hanya akan ada satu elemen yang diterima. Sedangkan jika mencari berdasarkan class, maka bisa saja akan terdapat lebih dari satu elemen yang diterima sehingga. Untuk mendapatkan konten dari elemen tersebut dapat digunakan method value().

Terdapat beberapa langkah-langkah yang akan dilakukan untuk melakukan *web scraping* ini. Langkah-langkahnya adalah sebagai berikut.

- Mengunduh data konten dari sebuah laman dengan format HTML.
- Lakukan algoritma *patern matching* untuk mencari apakah terdapat *patern* dalam sebuah teks. Jika ada, maka akan dilakukan *parsing* HTML umtuk mendapatkan elemennya.
- Jika ditemukan pola dengan menggunakan algoritma *patern matchin,* maka kembalikan string yang merupakan elemen dengan tag yang dicari.

Pada kelas scraper yang penulis buat, terdapat sebuah atribut yang merupakan isi kode HTML yang akan disimpan. Kemudian terdapat beberapa method penting seperti `getElementbyId(param)`, `getElementbyClass(param)`, dan `value()`. Pada getElementbyId, keluaran yang dihasilkan adalah *string*. Sedangkan pada getElementbyClass, keluaran yang dihasilkan adalah *list of string*. Pada value, keluaran yang dihasilkan adalah konten dari suatu elemen HTML tersebut.

Method di atas menggunakan algoritma pencarian string Boyer-Moore. Jika ditemukan *patern* dalam suatu teks kode html, maka program akan mengelola kode html sedemikian rupa sehingga didapatkan keseluruhan elemen dari kode HTML yang memiliki *patern* yang sama.

Jika kita menjalankan getElementbyClass(“center), program akan mencari *patern* “ `class=”center”` ” pada kode HTML. Jika pada kode tersebut terdapat *patern* yang sesuai, maka program akan mencari substring yang merupakan elemen dengan nama atribut yang sesuai yaitu class=”center”.

Hal yang sama juga dilakukan saat program menjalankan fungsi getElementById(“konten”). Program akan mencari *patern* “ `id=”konten”` ” pada kode HTML. Jika pada kode tersebut terdapat *patern* yang sesuai, maka program akan mencari substring yang merupakan elemen dengan nama atribut yang sesuai yaitu id=”konten”. Bedanya hanya hasil yang dikeluarkan seperti yang sudah dijelaskan sebelumnya.

Dengan menggunakan contoh sintaks html yang hampir sama seperti sebelumnya dengan beberapa penambahan, penulis akan mencoba menjalankan beberapa fungsionalitas dari *web scraper* yang telah dibuat.

```
...
<body>
  <h1 class=”center”> Judul Laman </h1>
  <div id=”konten”>
    <p class=”center”> Isi laman </p>
    <div class=”share”>
      <p> Postingan ini telah dishare sebanyak
</p>
      <p id=”bykshare”> 200 kali </p>
    </div>
  </div>
</body>
...
```

1. Pengujian methode getElementbyId()

Perintah yang dijalankan pada main program adalah sebagai berikut.

```
s = Scraper(text)
print(s.getElementbyId("bykshare"))
```

Keluaran yang dihasilkan adalah sebagai berikut.

```
PS C:\Users\USER\Desktop> python Scraper.py
<p id="bykshare"> 200 kali </p>
```

Method ini juga sudah dapat menampilkan keluaran yang memiliki banyak baris atau banyak elemen yang enjadi anak dari elemen tersebut. Contohnya adalah seperti berikut.

```
s = Scraper(text)
print(s.getElementbyId("konten"))
```

Keluaran yang dihasilkan adalah sebagai berikut.

```
PS C:\Users\USER\Desktop> python Scraper.py
<div id="konten">
  <p class="center"> Isi laman </p>
  <div class="share">
    <p> Postingan ini telah dishare sebanyak </p>
    <p id="bykshare"> 200 kali </p>
  </div>
</div>
```

2. Pengujian method getElementbyClass()

Perintah yang dijalankan pada main program adalah sebagai berikut.

```
s = Scraper(text)
print(s.getElementbyClass("center"))
```

Keluaran yang dihasilkan adalah sebagai berikut.

```
PS C:\Users\USER\Desktop> python Scraper.py
['<h1 class="center"> Judul Laman </h1>', '<p class="center"> Isi lama
</p>']
```

Method ini akan menghasilkan list of string sebanyak dua buah string karena, pada struktur html yang kita scrap, terdapat dua buah elemen yang memiliki atribut class=”center”

3. Pengujian methode value()

Perintah yang dijalankan pada main program adalah sebagai berikut.

```
s = Scraper(text)
print(s.getElementbyId("konten").value())
```

Keluaran yang dihasilkan adalah sebagai berikut.

```
PS C:\Users\USER\Desktop> python Scraper.py

  <p class="center"> Isi laman </p>
  <div class="share">
    <p> Postingan ini telah dishare sebanyak </p>
    <p id="bykshare"> 200 kali </p>
  </div>
```

Method ini akan menghasilkan konten dari suatu elemen saja. Sudah tidak ada lagi tag pembuka dan penutup untuk div dengan atribut id=”konten”.

B. Penerapan *Web Scraping* Untuk Mendapatkan Data Mahasiswa Teknik Informatika ITB

Dengan menggunakan *web scraper* yang lebih kompleks, misalnya *BeautifulSoup*, kita dapat lebih leluasa lagi dalam menggunakan *web scraping*. Misalnya adalah dalam proses mendapatkan data mahasiswa Teknik Informatika ITB.

Pada kesempatan kali ini, penulis akan mencoba untuk menerapkan algoritma *patern matching* dalam melakukan *web sraping* untuk mendapatkan data mahasiswa Teknik Informatika ITB melalui website forlapdikti (https://forlap.ristekdikti.go.id/mahasiswa/detailsemester/MkZBQkJBQUItOURCMy00MzMxLThFRTItNEFCNjZGRjVEMTU2/20181).

Setelah menganalisa bentuk dari struktur HTML laman forlap dikti, diperoleh bahwa data mahasiswa terletak dalam tag tr (table). Sehingga, dengan menggunakan *beautifulsoup* dan menggunakan method *find_all*, akan diperoleh konten-konten yang merupakan tag td (baris dari setiap kolom). Dengan menggunkan method *fin_all* lagi, akan diperoleh konten-konten nama mahasiswa dan NIM-nya berurutan berdasarkan nama.

Perintah yang dijalankan pada main program adalah sebagai berikut.

```
soup = BeautifulSoup(content, 'html.parser')
for data in soup.find_all('tr'):
    mhs = data.find_all('td')
    if (len(mhs) != 0):
        nim = mhs[1].text
        nama = mhs[2].text.title()
        print(nama, nim)
```

Keluaran yang dihasilkan adalah sebagai berikut.

```
PS C:\Users\USER\Desktop> python Scraper.py
Abda Shaffan Diva 13517021
Abdurrahman 13515024
Abdurrahman Adni 13517117
Abel Stanley 13517068
Abiyyu Avicena Ismunandar 13517083
Abner Adhiwijna 13516033
Abram Perdanaputra 13516083
Achmad Fahrurrozi Maskur 13515026
Adam Fadhel Ramadhan 13516054
Ade Surya Ramadhani 13514049
```

V. KESIMPULAN

Web scraping adalah proses mendapatkan data dengan cara mengekstrak struktur websitenya dengan menggunakan kode program. Ada banyak sekali kegunaan *web scraping.* Salah satu yang paling berguna adalah untuk mendapatkan data dari suatu laman yang tidak menyediakan *Application Programming Interface* (API) yang dapat memfasilitasi pertukaran informasi antar perangkat lunak.

Ada banyak teknik yang dapat digunakan dalam melakukan *web scraping*. Salah satu yang terbaik adalah HTML *Parsing*. Namun bukan berarti *web scraping* tidak dapat dilakukan dengan menggunakan *patern matching.* Walaupun terbatas, namun penulis mampu mengimplementasikan beberapa fungsionalitas dasar dalam melakukan web scraping.

Dengan *patern matching*, program akan mencari *patern* yang merupakan nama kelas atau id yang ingin dicari apakah ada eksistensinya pada kode program HTML. Jika ada, maka program akan memberikan elemen dari suatu kode HTML yang memiliki atribut yang sama dengan *patern*.

VI. UCAPAN TERIMA KASIH

Pertama-tama, penulis mengucapkan rasa syukur dan terima kasih untuk Allas SWT, Tuhan Yang Maha Esa atas limpahan syukur dan rahmatnya yang memberi kemudahan sehingga penuis masih dapat menempuh studi di Teknik Informatika Institut Teknologi Bandung.

Selanjutnya penulis mengucapkan terima kasih untuk kedua orang tua dan keluarga penulis yang selalu mendoakan dan mendukung kesuksesan penulis.

Penulis juga ingin mengucapkan banyak terima kasih atas limpahan ilmu yang senantiasa diberikan oleh Bapak Rinaldi Munir selaku dosen mata kuliah Strategi Algoritma yang sangat membantu penulis dalam penyusunan makalah ini.

Terakhir, penulis ingin mengucapkan terima kasih kepada teman-teman satu angkatan UNIX 2017 terkhusus teman-teman kelas K3 IF 2017 yang telah membantu perkembangan studi penulis termasuk dalam peyususnan makalah ini.

REFERENSI

[1] https://www.dewaweb.com/blog/web-scraping-panduan-dan-teknik-tekniknya/ diakses pada 25 April 2019 pukul 12.03

[2] https://developer.mozilla.org/id/docs/Learn/Getting_started_with_the_web/HTML_basics diakses pada 25 April 2019 pukul 13.09

[3] https://www.shieldsquare.com/what-are-the-different-scraping-techniques/ diakses pada 25 April 2019 pukul 14.23

[4] Munir, Rinaldi. 2004. Diktat Kuliah IF2211 Strategi Algoritma. Bandung: Program Studi Teknik Informatika Institut Teknologi Bandung.

[5] https://medium.com/@juniardiakbar/potensi-penyalahgunaan-data-mahasiswa-usu-5731f9a89b31 diakses pada 26 April 2019 08.07

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, 25 April 2019

Juniardi Akbar, 13517075