AN AUTOMATED ANALYTICS ENGINE FOR COLLEGE PROGRAM SELECTION USING MACHINE LEARNING AND BIG DATA ANALYSIS

Jinhui Yu, Xinyu Luan and Yu Sun

California State Polytechnic University, Pomona, CA, 91768

ABSTRACT

Because of the differences in the structure and content of each website, it is often difficult for international applicants to obtain the application information of each school in time. They need to spend a lot of time manually collecting and sorting information. Especially when the information of the school may be constantly updated, the information may become very inaccurate for international applicants. we designed a tool including three main steps to solve the problem: crawling links, processing web pages, and building my pages. In compiling languages, we mainly use Python and store the crawled data in JSON format [4]. In the process of crawling links, we mainly used beautiful soup to parse HTML and designed crawler.

In this paper, we use Python language to design a system. First, we use the crawler method to fetch all the links related to the admission information on the school's official website. Then we traverse these links, and use the noise_remove [5] method to process their corresponding page contents, so as to further narrow the scope of effective information and save these processed contents in the JSON files. Finally, we use the Flask framework to integrate these contents into my front-end page conveniently and efficiently, so that it has the complete function of integrating and displaying information.

KEYWORDS

Data Crawler, Data Processing, Web framework.

1. INTRODUCTION

International students will encounter many problems when they apply for American universities and collecting enrollment information is absolutely one of them. When we tried to find the admission information of several schools, we found that it would take me a lot of time to find information one by one. As far as we know, every year, millions of students apply to American universities from all over the world, which is with no doubt a significant part of American education system. To get accurate and complete information about their dream schools, students have to browse websites of lots of universities to seek for valuable information. However, the structures of these websites vary greatly, and majors can be categorized in totally different ways, which make it easy for students to get lost in the huge amount of information. Students' time is precious, and they should be able to get information they need easily and efficiently.

Although there are already some websites and apps to help international applicants integrate their information, these sites are unable to update information in time due to certain technical problems, which may adversely affect the application process. The data of these web pages is often manually entered when

programmers make web pages. Obviously, such a huge amount of data cannot be changed in time manually. This paper designs a tool to crawl the admissions data from various schools and visualize these information in a systematic and concise way which could help students shorten decision making time.

In terms of development language, PHP, Java and C are the mainstream programming languages, but they all have some problems. PHP is known as "the best language in the world" [6], but it has no concept of multithreading, not much support for asynchrony and so on, which are its shortcomings as a crawler: Although C language and C + + are the most efficient and performance languages, but the learning cost is very high, and the amount of code is too large. Java has a complete ecosystem and is the biggest competitor of python, but its code is very cumbersome, and the crawler needs to modify the code frequently.

Many methods can be used to parse the captured web pages; both beautifulsoup [7] and lxml [8] are important tools in web page parsing and information extraction. Lxml is a library written in Python, which can deal with XML and HTML quickly and flexibly. Although it has higher performance in dealing with HTML, its learning cost is relatively high, so it is not suitable for small projects.

In terms of web framework, Django is the most versatile Web development framework in the Python field, with complete functions, good maintainability and development speed, but it will be cumbersome for small project code. Tornado is a web server and web application framework written in Python language. Although it has fast processing speed, it has few plug-ins and is not very convenient and efficient.

My goal is to make a crawler program to get the admission information of major universities. Web crawler (also known as web spider) is a program that automatically grabs the information of the World Wide Web according to certain rules. There are some good methods we have used to build the system.

Firstly, we chose Python for development, because Python has high portability, and has a very powerful third-party library. The development based on this can greatly improve the development efficiency. And compared with Java and C language, Python code is more concise and easy to read and write.

Second, we use BeautifulSoup when we process a web page after crawling. BeautifulSoup is a Python library that fetches data from web pages. The advantage of BS is that, compared with Scrapy crawler framework, although the amount of code will be increased, it allows me to crawl information more flexibly and freely.

Third, in order to build the web to show the information we integrate, we chose the Flask framework, a lightweight and customizable web framework [1]. Flask has some obvious advantages. First of all, compared with Django and other similar frameworks, flask is more flexible, light, safe and easy to use. It allows me to complete the implementation of small and medium-sized websites or web services with rich functions in a short time. In addition, flask also has strong customization. we can add functions according to my own needs, and realize the enrichment and expansion of functions while keeping the core functions simple. Its powerful plug-in library allows me to realize personalized website customization and develop a powerful website.

The rest of the paper is organized as follows: Section 2 details the challenges we encountered during the experiment and design of the sample; Section 3 focuses on the details of the solution corresponding to the challenges we mentioned in Section 2; followed by section 4 which describes the related work and finally section 5 which provides a concluding comment and points out the future work of the project.

2. CHALLENGES

In order to build the tracking system, a few challenges have been identified as follows.

2.1. Finding Right Link to Use

The first problem we encountered was that when we started crawling with the admission page of a school's official website as the source page, we was frustrated that the results contained too many irrelevant URLs and even many external links. This is because each school's official website contains news, notices, events and other information. we need to determine which pages are related to application and enrollment, and crawl the links in these pages; and these pages sometimes contain some extra information. Links, the pages corresponding to these links are basically irrelevant to the application information, so we don't need to save these links. So we need to design a method to remove useless links and save all the links related to the application information so that we can crawl the content of the corresponding pages after these links.

2.2. Finding Out Useful Information

The second challenge that bothers me is that the crawled pages often include a lot of content, such as navigation bars, links, pictures, videos, texts, etc., all in the form of tags in the code. These irrelevant content are like "noise", making it difficult for me to quickly find the main content and information. How can we find the information we need from so many elements? When we crawled the data from the school's official website, we found that the code contains a lot of CSS and other elements. This part of the code is mainly used to modify the web page, such as adjusting the layout of the web page, font, color, pictures, etc., to make the display of the web page look better and regular, but they are not important. So we designed the method, we first remove some common modifier tags, so that we can locate the key tags and the text content in them more quickly. we then determine the position of the content block by calculating the number of tags, so as to obtain the main content of the page.

3. SOLUTION

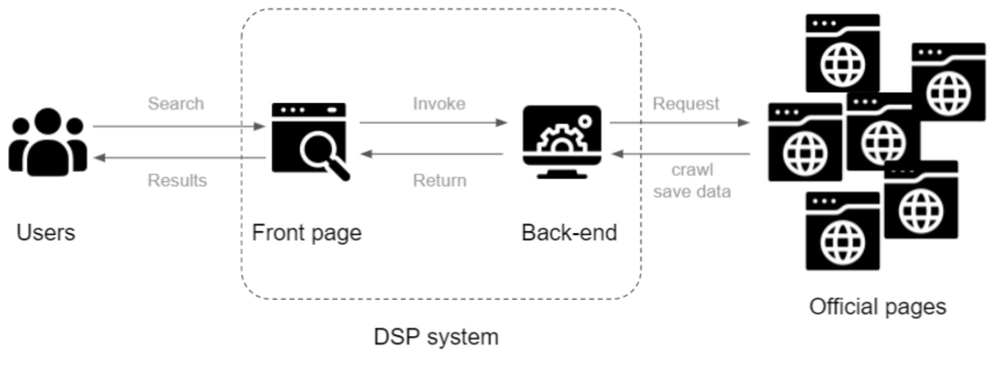


Figure 1. The overview of DSP system

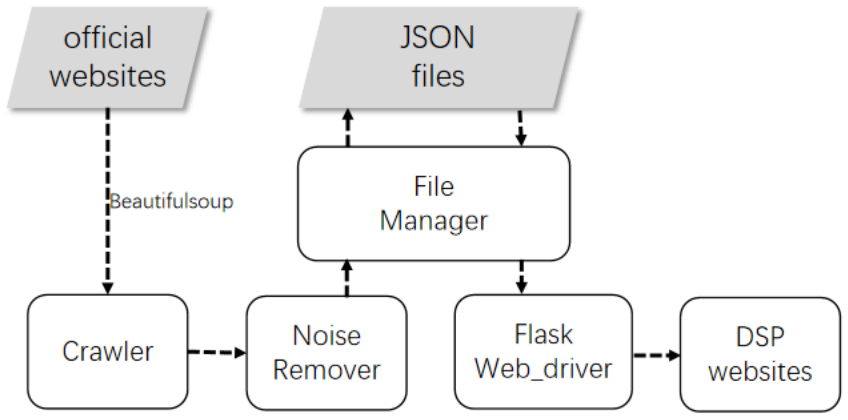


Figure 2. The o

DSP is a system based on crawler technology to assist students to apply the most well-known universities in the United States. DSP integrates the information of those programs. Users can browse the important information including university ranking, application conditions, department courses and so on; At the same time, users can also search for the Enrollment Requirements of various colleges and universities by searching professional names and courses, so as to quickly prepare for applying for universities. In DSP, we first crawl and save the enrollment information on the official websites of colleges and universities or some American authoritative websites, then filter, classify and integrate these information and data, and finally present them on our website in a more concise and easy way. Therefore, DSP can present the enrollment information of hundreds of well-known universities in the United States. Because all the information in DSP is obtained by crawling the official websites of universities, it can basically keep pace with the official websites of universities and update in real time, which means it will not provide applicants with expired or wrong information. Because the data processed by DSP will only contain enrollment information, applicants can use it to easily and quickly find the admission requirements, application deadline, tuition fees and other application related information of their favorite colleges, without being disturbed by other irrelevant content. The main technical difficulty of implementing a DSP system is how to automatically crawl and classify the enrollment information, since the structure of different official websites is very different. In addition, the processed data must be put on the front page in a more unified way, which is convenient for applicants to query. To achieve these goals, our tool consists of 5 main components (see the thick boxes in Figure 2):

Crawler

The most important goal of the whole DSP function is that we need to obtain all the links in the application or admission pages of the official websites of various schools, and store them in a container, so that we can crawl the contents of the corresponding pages of these links in the next step. In the process of crawling pages, DSP needs to solve two problems: duplicated pages removal and ensuring crawling within a certain range.

In the early process of crawling pages [9], we found that the links of a web page in a website had a loop. For example, we could see the link of the home page in the home page of the website, and then we might see a link to the home page in the sub-pages, and perhaps the sub-pages would also have corresponding links to the home page. Such crawling will lead to repeated crawling of the web page. To solve this problem, we need to design a class called Noise_remover to remove the duplicated pages.

In this class, my major idea was using Breadth First Search (BFS)[10] to crawl the URL and then processing it. In BFS, we need to use a data structure called queue. Queue [11] is a linear storage table, in which the element data is inserted at one end of the table and deleted at the other end, thus forming a FIFO (First In First Out) [12] table. So we defined a queue to_be_visited_pages_queue) to store the URLs to be crawled, and a set(visited_page_set) to store the crawled URLs. Set is a mutable container in Python whose data is not repeatable.

In Breadth First Search, we took the first element in the queue to crawl, then put all the URLs in the next layer at the end of the queue, and saved the visited URLs in the set. So in the next round of crawling, we could judge whether the URL was already visited_ page_ set to remove pages which we had already crawled.

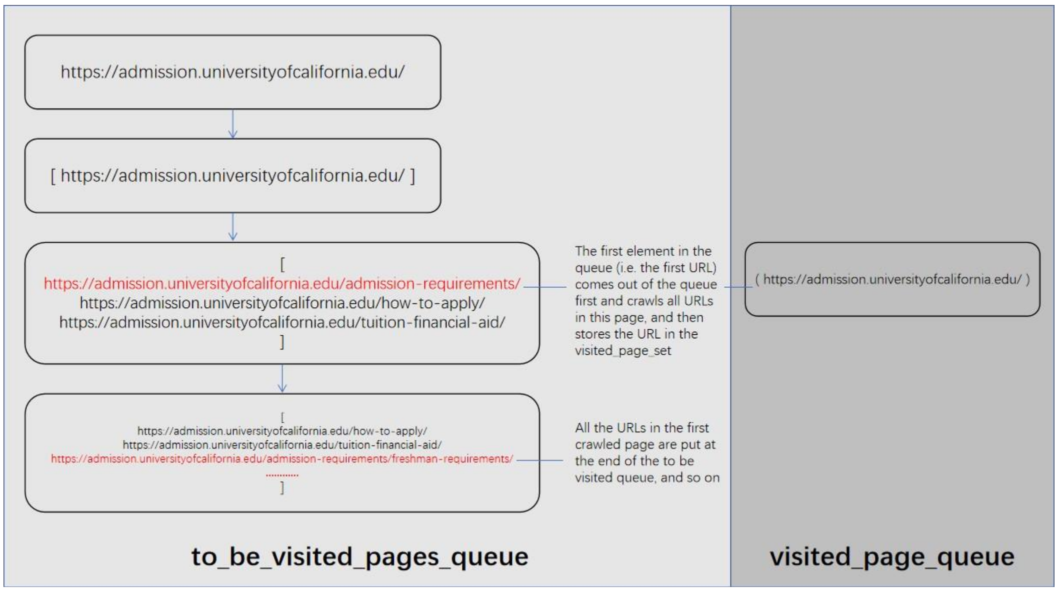
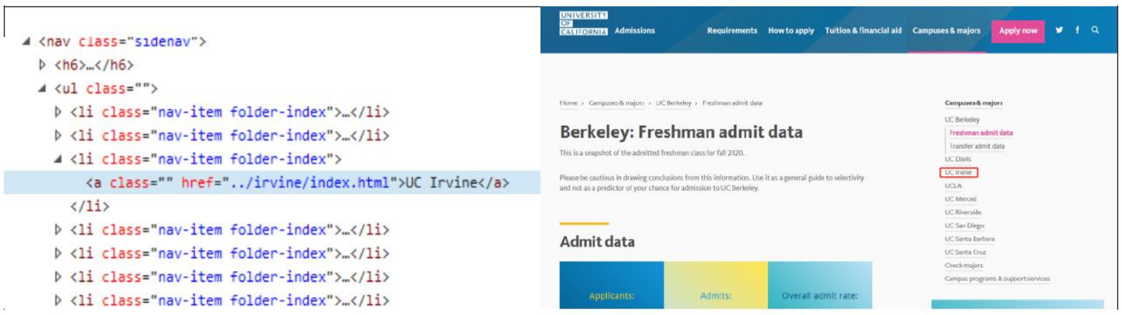
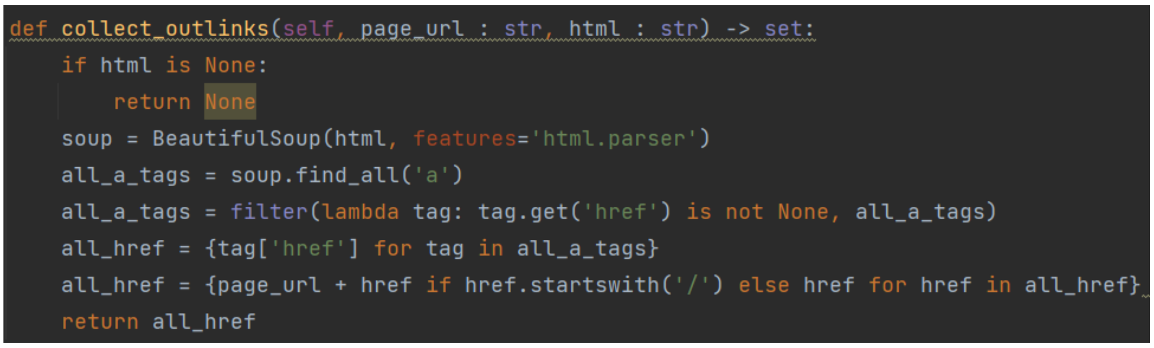

https://admission.universityofcalifornia.edu/
[https://admission.universityofcalifornia.edu/]
[https://admission.universityofcalifornia.edu/admission-requirements/ https://admission.universityofcalifornia.edu/how-to-apply/ https://admission.universityofcalifornia.edu/tuition-financial-aid/]
The first element in the queue (i.e. the first URL) comes out of the queue first and crawls all URLs in this page, and then stores the URL in the visited_page_set
(https://admission.universityofcalifornia.edu/)
[https://admission.universityofcalifornia.edu/how-to-apply/ https://admission.universityofcalifornia.edu/tuition-financial-aid/ https://admission.universityofcalifornia.edu/admission-requirements/freshman-requirements/ ....]
All the URLs in the first crawled page are put at the end of the to be visited queue, and so on
to_be_visited_pages_queue
visited_page_queue

Figure 3. The o

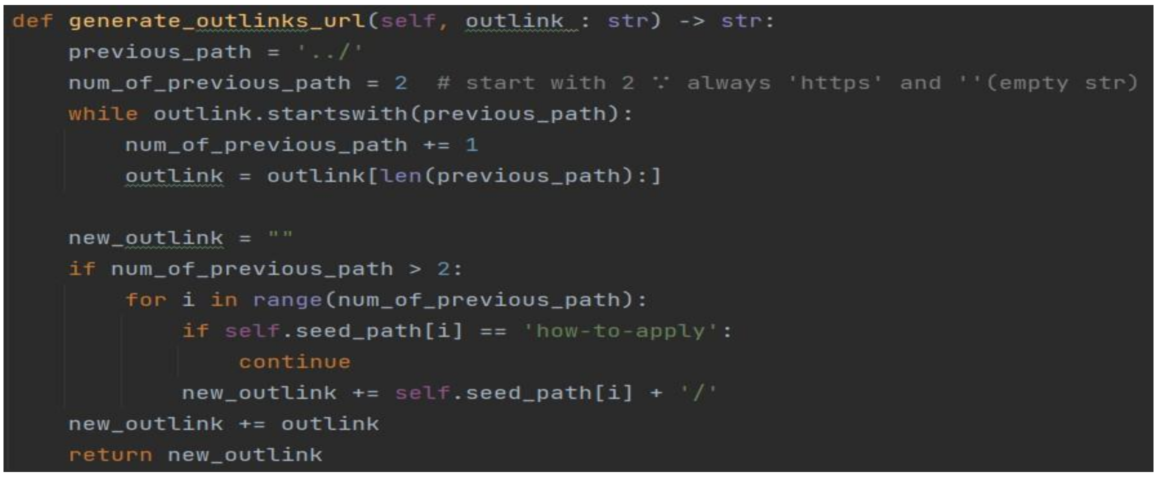
In addition, when we get the links in the page, we find that many of these links do not appear as complete links, but in the form of "../irvine/index/html". So we must complete their links before crawling.


```
◢ <nav class="sidenav">
  ▷ <h6>…</h6>
  ◢ <ul class="">
    ▷ <li class="nav-item folder-index">…</li>
    ▷ <li class="nav-item folder-index">…</li>
    ◢ <li class="nav-item folder-index">
        <a class="" href="../irvine/index.html">UC Irvine</a>
      </li>
    ▷ <li class="nav-item folder-index">…</li>
    ▷ <li class="nav-item folder-index">…</li>
    ▷ <li class="nav-item folder-index">…</li>
    ▷ <li class="nav-item folder-index">…</li>
```

I first used the function called collect_outlinks to get all URLs in a page, no matter whether they were complete or not. Then, we used the function called generate_outlinks_url to judge whether the URL was complete or not. If the URL was not complete, it would be automatically completed.



```python
def collect_outlinks(self, page_url : str, html : str) -> set:
    if html is None:
        return None
    soup = BeautifulSoup(html, features='html.parser')
    all_a_tags = soup.find_all('a')
    all_a_tags = filter(lambda tag: tag.get('href') is not None, all_a_tags)
    all_href = {tag['href'] for tag in all_a_tags}
    all_href = {page_url + href if href.startswith('/') else href for href in all_href}
    return all_href
```

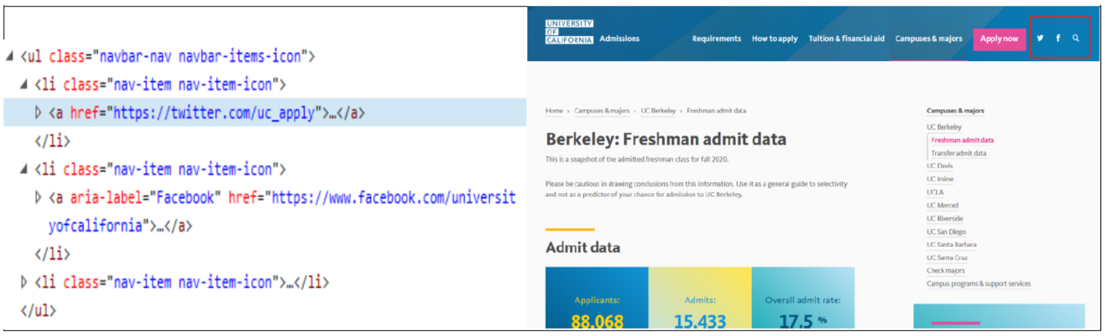


```python
def generate_outlinks_url(self, outlink : str) -> str:
    previous_path = '../'
    num_of_previous_path = 2  # start with 2 ∵ always 'https' and ''(empty str)
    while outlink.startswith(previous_path):
        num_of_previous_path += 1
        outlink = outlink[len(previous_path):]

    new_outlink = ""
    if num_of_previous_path > 2:
        for i in range(num_of_previous_path):
            if self.seed_path[i] == 'how-to-apply':
                continue
            new_outlink += self.seed_path[i] + '/'
    new_outlink += outlink
    return new_outlink
```

Figure 4. The o

There was still a small issue to be addressed. When crawling the school websites, we hope all the URLs are the internal pages of the schools, not some external links. As shown in the figure below, the page contains some links to Twitter and Facebook, which we need to remove as well.



```html
<ul class="navbar-nav navbar-items-icon">
  <li class="nav-item nav-item-icon">
    <a href="https://twitter.com/uc_apply">…</a>
  </li>
  <li class="nav-item nav-item-icon">
    <a aria-label="Facebook" href="https://www.facebook.com/universit
    yofcalifornia">…</a>
  </li>
  <li class="nav-item nav-item-icon">…</li>
</ul>
```

Berkeley: Freshman admit data

This is a snapshot of the admitted freshman class for fall 2020.

Please be cautious in drawing conclusions from this information. Use it as a general guide to selectivity and not as a predictor of your chance for admission to UC Berkeley.

Admit data

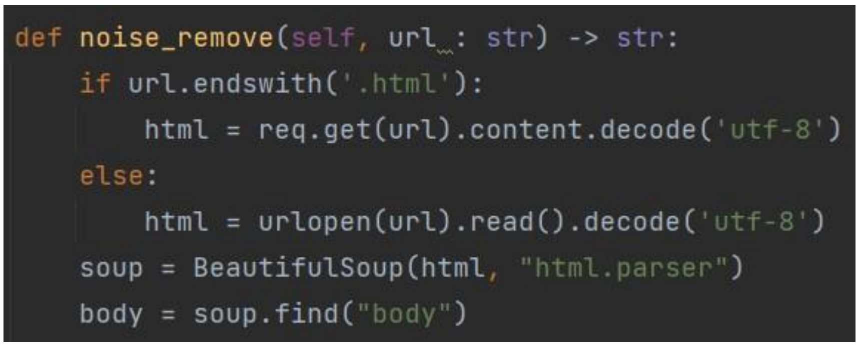
Applicants:	Admits:	Overall admit rate:
88,068	15,433	17.5 %

Figure 5. The o

Noise_ remover

After fetching all the links to admission related pages, we need to get more information about the page and the application, so we designed a class called Noise_ remover.

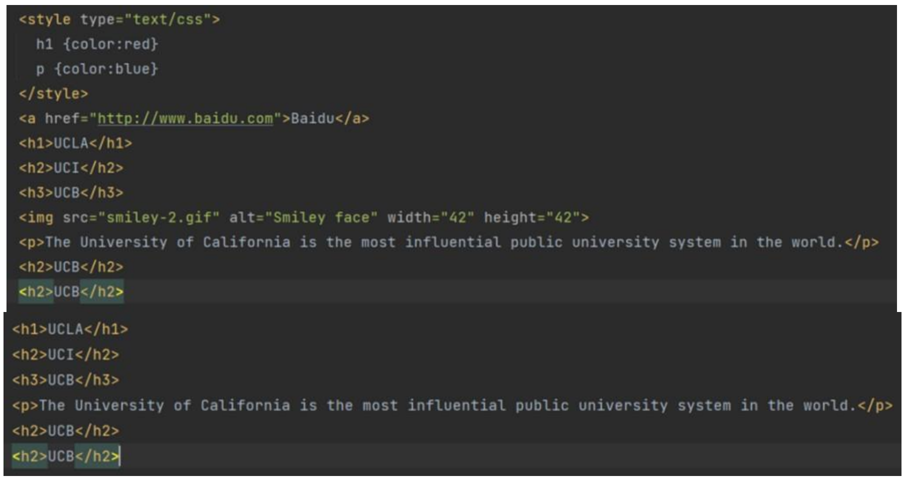
In Noise_ remover, we mainly use the noise_ remove function. This function first obtained the web page content through requests generating HTML data, and then used Beautifulsoup to parse the HTML data. Beatifulsoup is a kind of HTML parser, which uses knowledge of the syntax of the markup language to identify the structure [1]. At the same time, it provides easy-to-use navigation, search and modification operations, which greatly saves my programming time.



```python
def noise_remove(self, url : str) -> str:
    if url.endswith('.html'):
        html = req.get(url).content.decode('utf-8')
    else:
        html = urlopen(url).read().decode('utf-8')
    soup = BeautifulSoup(html, "html.parser")
    body = soup.find("body")
```

Figure 6. The o

After generating the beautiful soup object, we first need to remove the tags used to decorate the web page, such as CSS. we find the body tag through the built-in find method of beautiful soup, and then use the select function to find some of the CSS tags, such as body. Select ("image"), and then use the extract function to remove the modificatory content, so that only the main content of a page is left in the body tag.

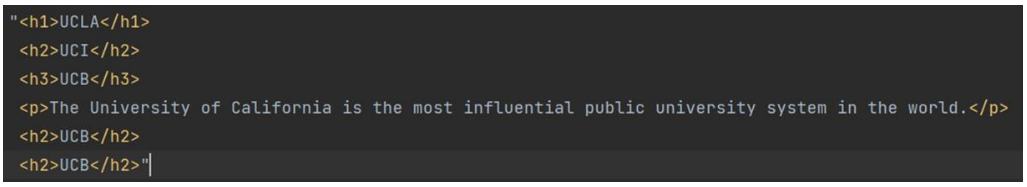


```html
<style type="text/css">
  h1 {color:red}
  p {color:blue}
</style>
<a href="http://www.baidu.com">Baidu</a>
<h1>UCLA</h1>
<h2>UCI</h2>
<h3>UCB</h3>
<img src="smiley-2.gif" alt="Smiley face" width="42" height="42">
<p>The University of California is the most influential public university system in the world.</p>
<h2>UCB</h2>
<h2>UCB</h2>
```

```html
<h1>UCLA</h1>
<h2>UCI</h2>
<h3>UCB</h3>
<p>The University of California is the most influential public university system in the world.</p>
<h2>UCB</h2>
<h2>UCB</h2>
```

Figure 7. Before After

I imported NLTK package [13] (Natural Language Toolkit), which is a python library commonly used in the NLP research field. First, we use str () to convert the content of <body> from BS4. Element. Tag type to string type; Next, we use word_ tokenize function to cut the string into words one by one.



```
"<h1>UCLA</h1>
 <h2>UCI</h2>
 <h3>UCB</h3>
 <p>The University of California is the most influential public university system in the world.</p>
 <h2>UCB</h2>
 <h2>UCB</h2>"
```

```
['<', 'h1', '>', 'UCLA', '<', '/h1', '>', '<', 'h2', '>', 'UCI', '<', '/h2', '>', '<', 'h3', '>',
'UCB', '<', '/h3', '>', '<', 'p', '>', 'The', 'University', 'of', 'California', 'is', 'the',
'most', 'influential', 'public', 'university', 'system', 'in', 'the', 'world.', '<', '/p', '>',
'<', 'h2', '>', 'UCB', '<', '/h2', '>', '<', 'h2', '>', 'UCB', '<', '/h2', '>']
```

Figure 8. The o

Then we designed a customize_ tokenizer function, traverses the cut content, and reassembles "<" and ">" and the content between them into a content list.

```
['<h1>', 'UCLA', '</h1>', '<h2>', 'UCI', '</h2>', '<h3>', 'UCB', '</h3>', '<p>',
'The', 'University', 'of', 'California', 'is', 'the', 'most', 'influential', 'public',
'university', 'system', 'in', 'the', 'world.', '</p>', '<h2>', 'UCB', '</h2>', '<h2>',
'UCB', '</h2>']
```

Figure 9. The o

Now, there are still a lot of useless tags in the list. To find content blocks, we count the amount of tags to analyze cumulative distribution of tags in the targeted pages.

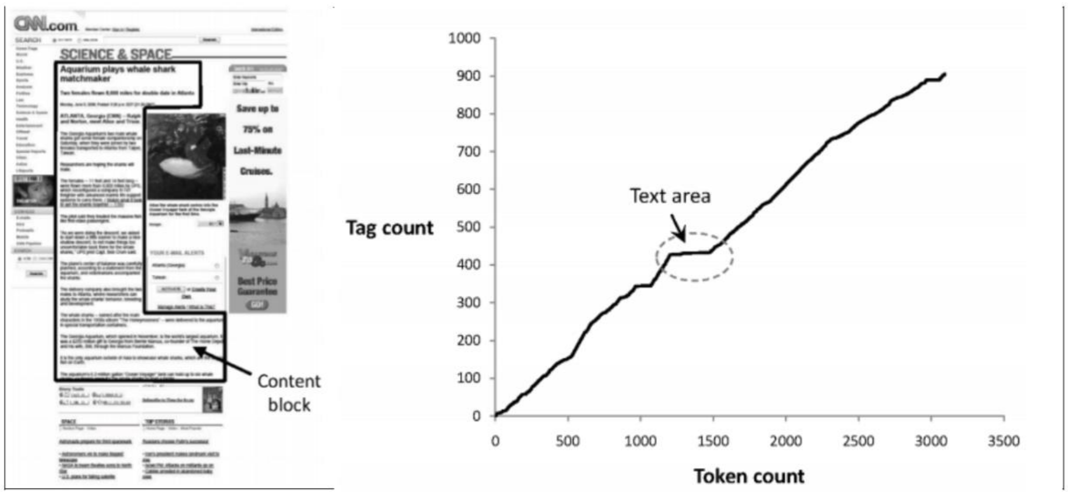


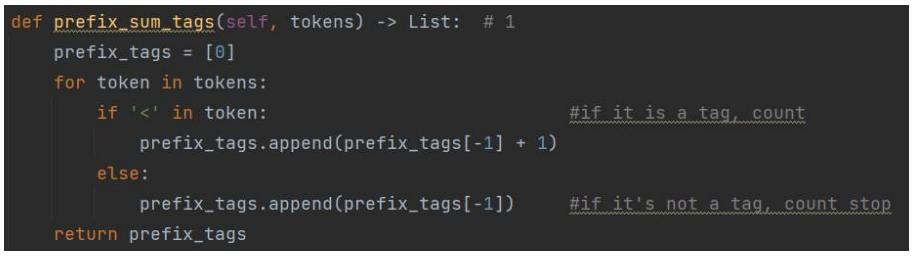
Figure 10. Noise Remove [2]

Now, there are still a lot of useless tags in the list. To find content blocks, we need to count the amount of tags to analyze the cumulative distribution of tags in the targeted pages, and the main text content of the page corresponds to the “plateau” in the middle of the distribution. This flat area is relatively small because of the large amount of formatting and presentation information in the HTML source for the page[2]. So we design a method to determine where the “plateau” is. Represent a web page as a sequence of bits, where bn = 1 indicates that the nth element in the content list is a tag. The values of we and j which maximize both the number of tags below we and above j and the number of non-tag tokens between we and j can help me find the position of main text content. we designed a function called prefix_ sum_ Tags , in which we use a new list to count.

i.e., maximize

$$\sum_{n=0}^{i-1} b_n + \sum_{n=i}^{j}(1-b_n) + \sum_{n=j+1}^{N-1} b_n$$

Figure 11. The o



```python
def prefix_sum_tags(self, tokens) -> List:  # 1
    prefix_tags = [0]
    for token in tokens:
        if '<' in token:                        #if it is a tag, count
            prefix_tags.append(prefix_tags[-1] + 1)
        else:
            prefix_tags.append(prefix_tags[-1])     #if it's not a tag, count stop
    return prefix_tags
```

Figure 12: The o

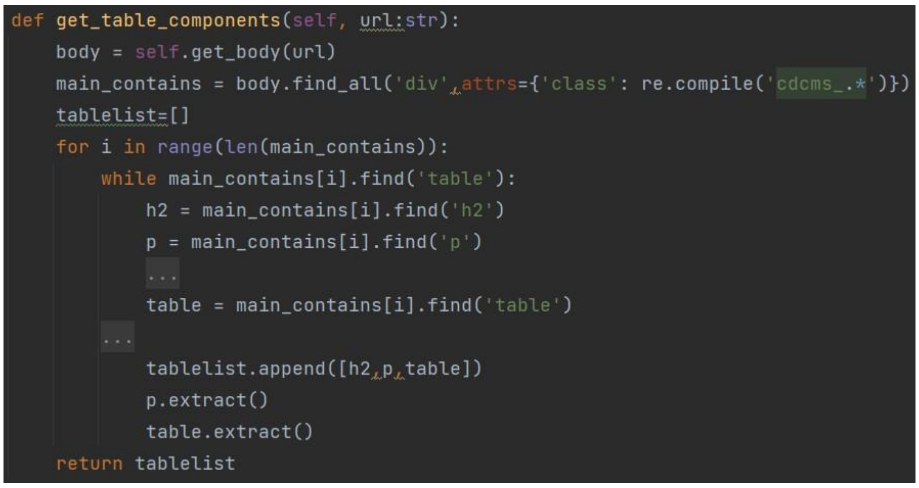
```
['<h1>', 'UCLA', '</h1>', '<h2>', 'UCI', '</h2>', '<h3>', 'UCB', '</h3>', '<p>',
'The', 'University', 'of', 'California', 'is', 'the', 'most', 'influential', 'public',
'university', 'system', 'in', 'the', 'world.', '</p>', '<h2>', 'UCB', '</h2>', '<h2>',
'UCB', '</h2>']
```

```
[0,1,1,2,3,3,4,5,5,6,7,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,9,10,10,11,12,12,13]
```

Figure 13. The o

Retrieve Table

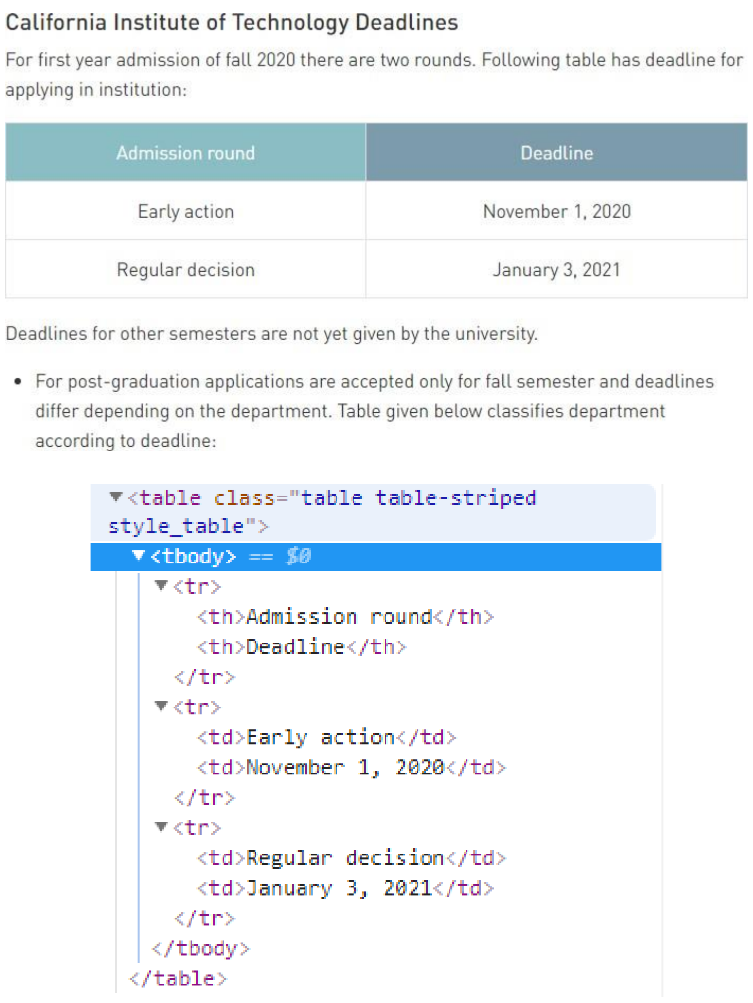
In order to get some tables on the page, we first used get_ table_ components method, then used find ('table ') to put all the tables in the page into a list.



```python
def get_table_components(self, url:str):
    body = self.get_body(url)
    main_contains = body.find_all('div',attrs={'class': re.compile('cdcms_.*')})
    tablelist=[]
    for i in range(len(main_contains)):
        while main_contains[i].find('table'):
            h2 = main_contains[i].find('h2')
            p = main_contains[i].find('p')
            ...
            table = main_contains[i].find('table')
        ...
            tablelist.append([h2,p,table])
            p.extract()
            table.extract()
    return tablelist
```

Figure 14. The o

Because most of the tables are made with HTML tags such as < tr > < td >, we chose to save these tables by row. we first use find ('tr') to determine the rows of the table, and then use find ('td') or find ('th') to take out all the elements of each row and put them into a list. Finally, we put all the lists representing rows into a list representing table, and we get a two dimensional list containing all the information of this table, because values in the list can be obtained through index, it is very convenient for me to read the information in these tables.



California Institute of Technology Deadlines

For first year admission of fall 2020 there are two rounds. Following table has deadline for applying in institution:

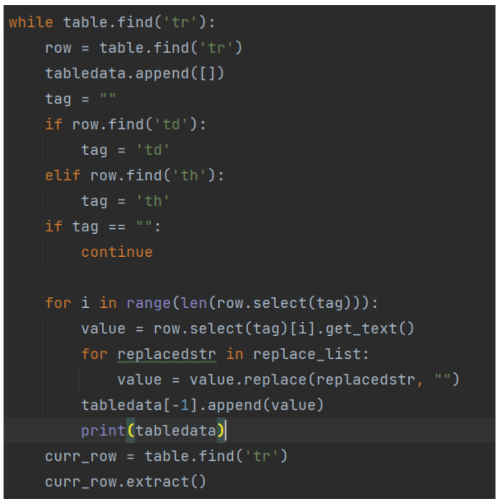
Admission round	Deadline
Early action	November 1, 2020
Regular decision	January 3, 2021

Deadlines for other semesters are not yet given by the university.

- For post-graduation applications are accepted only for fall semester and deadlines differ depending on the department. Table given below classifies department according to deadline:

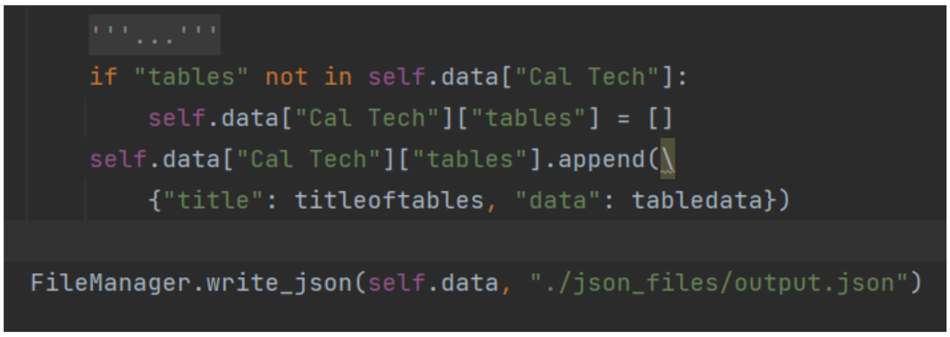
```
▼<table class="table table-striped
style_table">
  ▼<tbody> == $0
    ▼<tr>
        <th>Admission round</th>
        <th>Deadline</th>
      </tr>
    ▼<tr>
        <td>Early action</td>
        <td>November 1, 2020</td>
      </tr>
    ▼<tr>
        <td>Regular decision</td>
        <td>January 3, 2021</td>
      </tr>
    </tbody>
  </table>
```

Figure 15. The code excerpt for the forntend web pages



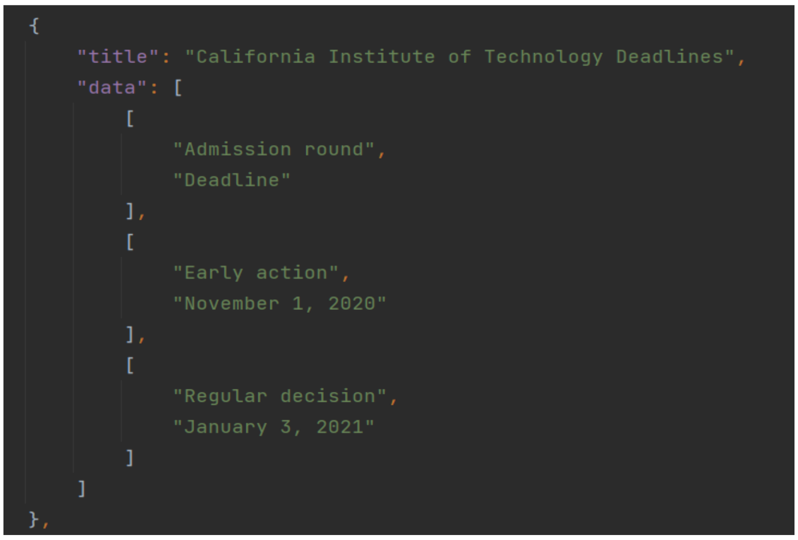
```
while table.find('tr'):
    row = table.find('tr')
    tabledata.append([])
    tag = ""
    if row.find('td'):
        tag = 'td'
    elif row.find('th'):
        tag = 'th'
    if tag == "":
        continue

    for i in range(len(row.select(tag))):
        value = row.select(tag)[i].get_text()
        for replacedstr in replace_list:
            value = value.replace(replacedstr, "")
        tabledata[-1].append(value)
        print(tabledata)
    curr_row = table.find('tr')
    curr_row.extract()
```



```
    '''...'''
    if "tables" not in self.data["Cal Tech"]:
        self.data["Cal Tech"]["tables"] = []
    self.data["Cal Tech"]["tables"].append(\
        {"title": titleoftables, "data": tabledata})

FileManager.write_json(self.data, "./json_files/output.json")
```



```
{
    "title": "California Institute of Technology Deadlines",
    "data": [
        [
            "Admission round",
            "Deadline"
        ],
        [
            "Early action",
            "November 1, 2020"
        ],
        [
            "Regular decision",
            "January 3, 2021"
        ]
    ]
},
```

Figure 16. The code excerpt for JSON data

4. Related Work

Stella and Woodhouse discussed the issues related with public ranking for higher education institutions [14]. They pointed out that the ranking may not effectively reflect the true quality of the institutions, which aligns the purpose of our work. However, they did not use any big data to draw the conclusion. Our approach focuses on using real school data and automated framework to get a more objective result.

The researchers Aguillo and Orduna-Malea discussed how they make a development of the "Would Class University and The ranking web"[15]. In their research paper, they indicate that the web application can assess most of the top universities by using G-factor which is a statistical factor that captures the diversity of motivations. However, unlike doing web crawling and noise removal in our application, their application doesn't do open source updates from the ranking universities.

Gupta and Divakar provided an imported approach to ranking web documents in 2012. Their ranking improved from their excremental data from the overall search results [16]. The ordering process will utilize the previous ranking score and train the new data with the previous data to make the new ranking more credible. However, their ranking is mostly finished with HTML only, but no framework like flask that our application used to make the web app more manageable and flexible.

5. CONCLUSIONS AND FUTURE WORK

The purpose of DSP application is to create an information integration center for students to reduce the complexity of applying to undergraduate or graduate university. In the web application, we included top rank university information for admission, so that students don't need to search by themselves to collect admission requirements.

To retrieve information from the official university website, we applied Web crawling to scrap the amount of weblinks. During the Web crawling process, we used BFS with a data structure queue to traverse web outlinks. Beautiful soup, a tool based on pythons, can help me to get all outlinks from root links and do basic parsing. we utilized the algorithm of noise removal to solve the problem of obtaining all the main contents and eliminating the useless information, such as ads, menu bar, header, footer etc. In order to make my application understandable and manageable, we used the flask framework to build the skeleton of the applications. In addition, we used Python as the main development language, because all the techniques we am using, like beautiful soup and flask, fit with Python. HTML, CSS, and JavaScript is used for the frontend of my web application. To create tables to integrate the information that we retrieved from information retrieval [3]. we used jQuery to make the creating table function reusable with all various tables and render that with HTML.

My application is deployed with Heroku.com. A couple of users and my friends viewed my website, and the feedback is good so far. The next step for developing the application is to create a community for users to char or share their experience. The final goal of the applications should be a community with full functionality for university admission.

Although this tool can effectively extract information for most of the application information pages composed of text and simple forms, there are still some pages made of pictures or more complex forms, and the extraction of this part of information may not be accurate and complete.

The function of this tool is relatively simple. If applicants can select institutions according to their scores and background, the application efficiency will be further improved.

Next, we hope to continue to develop new functions for this tool, such as the function of screening and matching schools, and improve the existing code to improve the applicability and ensure that all forms of information can be crawled. In addition, we will improve my algorithm to further speed up the crawling progress, so that this tool can maximize its advantages.

REFERENCES

[1] M. Grinberg, Flask web development: Developing advanced web applications with python. Sebastopol, CA: O'Reilly Media, 2014.
[2] W. B. Croft, D. Metzler, and T. Strohman, Search engines: Information retrieval in practice. Addison-Wesley Professional, 2011.
[3] "Information Retrieval Systems," Sciencedirect.com. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/information-retrieval-systems. [Accessed: 23-Jul-2021].
[4] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta, "Comparison of JSON and XML data interchange formats: A case study," Montana.edu. [Online]. Available: https://www.cs.montana.edu/izurieta/pubs/caine2009.pdf. [Accessed: 23-Jul-2021].
[5] H. K. Azad, R. Raj, R. Kumar, H. Ranjan, K. Abhishek, and M. P. Singh, "Removal of noisy information in web pages," in Proceedings of the 2014 International Conference on Information and Communication Technology for Competitive Strategies - ICTCS '14, 2014.
[6] W. Cui, L. Huang, L. Liang, and J. Li, "The research of PHP development framework based on MVC pattern," in 2009 Fourth International Conference on Computer Sciences and Convergence Information Technology, 2009, pp. 947–949.
[7] "Beautiful Soup Documentation — Beautiful Soup 4.9.0 documentation," Crummy.com. [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/. [Accessed: 23-Jul-2021].
[8] "Lxml - processing XML and HTML with Python," Lxml.de. [Online]. Available: https://lxml.de/. [Accessed: 23-Jul-2021].
[9] "How Google's site crawlers index your site - Google search," Google.com. [Online]. Available: https://www.google.com/search/howsearchworks/crawling-indexing/. [Accessed: 23-Jul-2021].
[10] S. Beamer, K. Asanovic, and D. Patterson, "Direction-optimizing breadth-first search," in 2012 International Conference for High Performance Computing, Networking, Storage and Analysis, 2012, pp. 1–10.
[11] G. L. Hajba, Website scraping with python: Using BeautifulSoup and scrapy, 1st ed. Berlin, Germany: APress, 2018.
[12] "FILO," Techterms.com. [Online]. Available: https://techterms.com/definition/filo. [Accessed: 23-Jul- 2021].
[13] E. Loper and S. Bird, "NLTK: The Natural Language Toolkit," arXiv [cs.CL], 2002.
[14] "Ranking of higher education institutions / Antony Stella and David Woodhouse," Gov.au. [Online]. Available: https://catalogue.nla.gov.au/Record/5784960. [Accessed: 22-Jul-2021].
[15] I. F. Aguillo and E. Orduña-Malea, "The ranking web and the 'world-class' universities," in Building World-Class Universities, Rotterdam: SensePublishers, 2013, pp. 197–217.
[16] P. Gupta, S. K. Singh, D. Yadav, and A. K. Sharma, "An improved approach to ranking web documents," J. Inf. Process. Syst., vol. 9, no. 2, pp. 217–236, 2013.