Adaptive Citation Extraction with Regex and String Matching for Multiple Source Types and Auto-Conversion

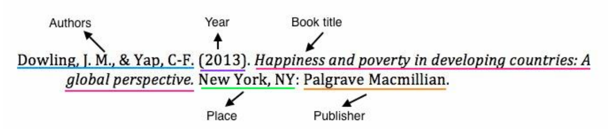
Heleni Gratia Meitrina Tampubolon - 13523107[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]helenigratia@gmail.com, 13523107@std.stei.itb.ac.id

Abstract— **Creating citations is an important part of academic writing, but many existing tools require users to manually choose the type of source and follow strict input formats. This paper presents a simple and adaptive method to extract citation details using concept of string matching, that is regular expressions (regex). The system can automatically find key information such as author names, publication dates, DOIs, and journal titles from different kinds of sources, like research articles, news websites, and blogs. After collecting the data, it can turn it into proper citation formats like APA, MLA, or Chicago using a flexible design. This method helps make citation easier and faster, especially when working with many links from different websites.**

***Keywords*—Citation, Regex, Pattern Matching**

I. Introduction

Citations are written references to sources of information used in academic or other scholarly works. These sources can include journal articles, books, websites, or various other forms of media. A citation provides a concise description of the source, typically including crucial information such as the author's name, work title, journal or publisher, volume, publication date, and other identifying details.



Dowling, J. M., & Yap, C-F. (2013). *Happiness and poverty in developing countries: A global perspective.* New York, NY: Palgrave Macmillian.

Figure 1.1. Citation Structure
Source: [1]

The primary function of citation is to acknowledge the intellectual contributions of previous authors or researchers. Moreover, citations ensure the absence of plagiarism by clearly indicating which parts of a text are derived from or adapted from other sources. This practice also allows readers to easily trace and verify the information presented, helps make the work more trustworthy and believable. The importance of citations becomes especially evident in writing academic papers or scholarly publications, given that citations form a cornerstone of academic integrity.

Given the sheer volume of academic literature and the increasing complexity of research, the manual process of identifying, extracting, and standardizing citations can be exceedingly time-consuming and prone to human error. Automating citation extraction becomes very important as it greatly reduces the manual work for researchers and writers. An automated system can quickly scan large amounts of text, find citation patterns, and pull out important details faster and more consistently than people can. This not only makes the process of literature review and reference management easier but also ensures better accuracy and following of different citation styles, allowing users to spend more time on writing content instead of boring administrative tasks.

To facilitate citation management, various tools are available, both as websites and applications, including Mendeley, Quillbot, and even built-in features within Google Docs. Within the academic world, there are also various standardized citation styles, such as MLA (Modern Language Association), APA (American Psychological Association), and Chicago Manual of Style, each with its own specific formatting rules.

Despite the availability of these tools and citation styles, the process of extracting citations from free-form text still poses a significant challenge, particularly when faced with diverse formatting. Therefore, this paper focuses on adaptive citation extraction using a combination of regular expressions (regex) and string matching. Regex is a sequence of characters that forms a search pattern, enabling the identification of complex and varied text patterns. This approach allows for a flexible and efficient pattern search process within texts, even for citations that lack strict structural uniformity.

Through the application of this method, this paper aims to offer preliminary insights and a practical basis for further exploration in the field of automated information extraction, particularly in reference and citation management. While it may not present a major breakthrough, the findings are intended to support the ongoing development of more accurate, user-friendly, and intelligent systems for automating scholarly writing and literature organization in the future.

II. Theoretical Framework

A. String Matching

String matching is a fundamental computational technique used to locate the presence and position of a specific pattern within a larger body of text. In formal terms, the problem can be defined as follows: given a text T of length n (where T is a long string), and a pattern P of length m (where m <<< n), the objective is to find the first occurrence of P in T [2]. That is, to determine the index or position in T where a substring exactly matches P. Various algorithms have been developed to perform string matching efficiently, ranging from brute-force approaches to more optimized methods such as Knuth-Morris-Pratt (KMP) and Boyer-Moore (BM) algorithms. These algorithms differ in terms of preprocessing, time complexity, and the strategy used to compare characters between T and P. Regardless of the method, the core goal remains the same, to detect exact matches of a pattern within a larger string as quickly and accurately as possible.

B. Regular Expression (Regex)

a. Definition

Regular expressions (regex) are sequences of characters that define specific search patterns. By combining literal characters with special symbols, regex allows for the precise specification of string patterns. This capability enables regex to match a wide variety of text structures, making it highly effective for tasks such as searching within documents, replacing specific patterns, validating user input, and parsing structured or unstructured data.

In the context of string matching, regex provides a more flexible and expressive approach compared to simple substring search. A regular expression can match one or multiple substrings within a larger text. For example, the pattern [a-z]+ matches any sequence of one or more lowercase letters, while the pattern \d{3} matches exactly three digits.

b. Regex Syntax

1) Character
All characters, except those having special meaning in regex, matches themselves. E.g., the regex x matches substring "x", regex 7 matches "7", regex = matches “=”, and regex @ matches "@".
2) Special Regex Characters
Certain characters have special functions in regex and are not interpreted as literal characters. E.g. ., +, *, ?, ^, $, (,), [,], {, }, |, and \.
3) Escape Sequences (\char)
Used to match a character that has special meaning in regex. A backslash \ prefixes the special character. Also used for common escape sequences. E.g., \. matches "."; regex \+ matches "+"; and regex \(matches "(".
4) A Sequence of Characters (or String)
Matches an exact sequence of characters and is case-sensitive by default. E.g. Regex Saturday matches the string "Saturday".
5) Character Classes (Bracket Lists)
Character classes allow matching any single character from a defined set.

Table 2.1. Character Classes Regex
Source: Personal Collection

Notation	Description
[abc]	Matches any one of the characters within the brackets.
[0-9] or [A-Za-Z]	Matches any one character within the specified range.
[^abc]	Matches not one of the characters within the brackets (negation).

6) Occurrence Indicators (or Repetition Operators)
These operators specify how many times the preceding element of the regex must occur.

Table 2.2. Occurrence Indicators Regex
Source: Personal Collection

Notation	Description
+	Matches one or more (1+) occurrences of the preceding element.
*	Matches zero or more (0+) occurrences of the preceding element. It includes the empty string if no match is found.
?	Matches zero or one (optional) occurrence of the preceding element.
{m,n}	Matches exactly m to n (inclusive) occurrences of the preceding element. E.g. a{2,4} matches "aa", "aaa", or "aaaa”
{m}	Matches exactly m occurrences of the preceding element. E.g. a{3} matches "aaa"
{m,}	Matches m or more (m+) occurrences of the preceding element.

7) Meta-characters
Meta-characters are shorthand notations for common character sets.

Table 2.3. Meta-characters Regex
Source: Personal Collection

Notation	Description
.	Matches any one character, except newline (\n).
\d	Matches any one digit character.
\D	Matches any one non-digit character.
\w	Matches any one "word" character (alphanumeric and underscore).
\W	Matches any one “non-word” character.
\s	Matches any one whitespace character.
\S	Matches any one non-whitespace character.

8) Position Anchors
Position anchors do not match characters but specific positions within the text, such as the beginning or end of a line or word.

Table 2.4. Position Anchors Regex
Source: Personal Collection

Notation	Description
^	Matches the start of a line.
$	Matches the end of a line.
\b	Matches a **word boundary** (the position between a word character and a non-word character, or the

	start/end of the string).
\B	Matches a non-word boundary.
\<	Matches the start of a word.
\>	Matches the end of a word.
\A	Matches the start of the entire input string.
\Z	Matches the end of the entire input string.

9) Parenthesized Back References

Parentheses () are used not only to group parts of a regular expression but also to create capturing groups. These groups store the matched text, which can be referenced later within the same expression or in replacement operations.

To define a capturing group, enclose the desired part of the pattern in parentheses.
E.g. The regex (ha)\1 matches the string "haha”, the \1 refers back to the text matched by the first group (ha).

c. Regex in Python

Python offers powerful support for regular expressions through the built-in re module. This module provides various functions and methods to work with regular expression patterns, enabling pattern matching, searching, substitution, and more. [3]

1. re.compile(pattern, flags=0)
   Compiles a regular expression pattern into a regex object. This object can then be reused to perform matching operations using methods like .match() and .search().
```
import re
pattern = re.compile(r"\d+")
```
2. pattern.search(string[, pos[, endpos]])
   This function to scan through the string to find the first location where the pattern matches. Returns a match object if found, or None otherwise.
```
result = pattern.search("Order #12345")
# Matches "12345"
```
3. pattern.match(string[, pos[, endpos]])
   This function attempts to match the regular expression pattern only at the beginning of the specified string. If a match is found, it returns a match object; otherwise, it returns None.
```
result  =  pattern.match("123abc")    #
Matches "123"
```
4. re.findall(pattern, string, flags=0)
   This function returns all non-overlapping matches of the pattern in the string as a list. If the pattern has capturing groups, it returns a list of tuples.
```
matches  =  re.findall(r"\d+",  "Item  1
costs 25 and item 2 costs 40")
# Output: ['1', '25', '2', '40']
```

C. Brute Force Algorithm

a. Definition

The brute force algorithm is a straightforward problem-solving approach that systematically examines all possible solutions to identify the correct one. In the context of pattern matching and string processing, brute force methods work by exhaustively checking every possible position or combination until the target pattern is found or all possibilities have been tested.

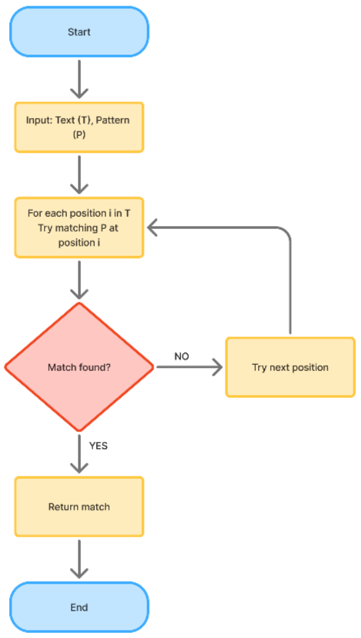

Start
Input: Text (T), Pattern (P)
For each position i in T Try matching P at position i
Match found?
NO
Try next position
YES
Return match
End

Figure 2.1. Flowchart Brute Force Regex
Source: Personal Collection

b. Characteristics [4]

1. Simpilicty and Directness
   The algorithm follows the most obvious and straightforward approach to problem solving. Rather than employing sophisticated optimization techniques, it systematically checks every possible solution candidate. This "just do it" philosophy ensures that the implementation remains clear and easily understandable, making it an excellent baseline approach for complex problems.
2. Completeness and Reliability
   One of the strongest advantages of brute force methods is their guarantee of finding a solution if one exists. By exhaustively examining all possibilities, these algorithms eliminate the risk of missing valid solutions due to heuristic shortcuts or optimization assumptions.
3. High Computational Complexity
   The trade-off for simplicity and completeness is typically high time and space complexity. Brute force algorithms often exhibit exponential or polynomial time complexity, making them impractical for large-scale problems without optimization.

III. Implementation

In this section, a demonstration of the regex method for adaptive citation is presented. The focus of this section is to demonstrate the implementation of the proposed system in handling citation extraction and formatting. The program is written in Python and designed to extract citation data using regex, covering various types of sources such as journal articles, news websites, and encyclopedia entries.

This implementation also includes adaptive regex handling based on source type, allowing the system to adjust its pattern matching depending on the structure of the input. Furthermore, it features an auto style conversion engine that supports multiple citation formats including APA, MLA, and Chicago. The system is also capable of processing multiple article URLs in a single batch, making it efficient for users dealing with large volumes of references.

A. Input Data

This program takes input from the command line interface (CLI), allowing the user to choose between using example URLs for citation extraction, entering their own URLs (multiple URLs can be separated by commas), or exiting the program. Users can enter any URL they want to extract citations from, including journal articles, news websites, and other online sources.

B. Preprocessing

In this implementation, the source is in the form of a URL. Therefore, the system must process the URL using HTTP requests to retrieve the web page content. This is done by sending a request through the requests library at python and parsing the response with BeautifulSoup to extract relevant metadata. The preprocessing step is handled in the extract_from_url() method of the CitationExtractor class.

Table 3.1. Function `extract_from_url`

```
  def extract_from_url(self, url: str) ->
CitationData:
          try:
              response = self.session.get(url,
timeout=10)
              response.raise_for_status()
              soup =
BeautifulSoup(response.content, 'html.parser')

              citation_data =
CitationData(url=url)
              citation_data.source_type =
self._detect_source_type(url, soup)
              citation_data.title =
self._extract_title(soup)
              citation_data.authors =
self._extract_authors(soup)
              citation_data.publication_date =
self._extract_date(soup)
              citation_data.publisher =
self._extract_publisher(soup)
              citation_data.journal =
self._extract_journal(soup)
              citation_data.doi =
self._extract_doi(soup)

              if citation_data.source_type ==
"academic":
                  citation_data.volume =
self._extract_volume(soup)
                  citation_data.issue =
self._extract_issue(soup)
                  citation_data.pages =
self._extract_pages(soup)

              return citation_data

          except Exception as e:
              print(f"Error extracting from URL
{url}: {str(e)}")
              return CitationData(url=url,
title="Unable to extract title")
```

This method starts by creating a request session with a user-agent header, which helps the program act like a regular web browser so it can access most websites. After getting the content from the webpage, the program reads (parses) the HTML to find important parts needed for citations, such as the title, author(s), date of publication, publisher, journal name, DOI, and if it is an academic source, also volume and issue. This part will related to other classes, Citation Extractor.

The system also checks what type of source it is. Whether it is from an academic journal, news website, book, or just a regular website by looking at the website address and some specific tags in the webpage. This preprocessing step helps the program adjust to different types of sources, so it can later create accurate citations in the chosen format, like MLA, APA, or Chicago.

C. Citation Extractor

1. Extract Title

The first step is to get the title from the webpage. The program looks for meta tags like citation_title, og:title, or DC.title. If not found, it tries to get the title from the <title> or <h1> tag. If none are found, it returns “No title found.” It helps the program get the title automatically from different types of websites.

2. Extract Authors

The next step is to get the author names. The program first looks for meta tags like citation_author, author, or DC.creator. If not found, it checks JSON-LD data in the page for author info. If those still don’t work, the program uses regular expressions (regex) to search for author patterns in the page text, such as “By [Name]” or “Written by [Name]”. This helps find author names from many types of websites, even when the structure is not standard.

Table 3.2. Author Patterns

```
  if not authors:
              author_patterns = [
                  r'By\s+([A-Z][a-z]+\s+[A-Z][a-
z]+)',
                  r'Author[s]?:\s*([^\\n]+)',
                  r'Written by\s+([A-Z][a-
z]+\s+[A-Z][a-z]+)'
              ]

              text = soup.get_text()
              for pattern in author_patterns:
                  matches = re.findall(pattern,
text)
                  authors.extend(matches)
```

3. Extract Date

The program first looks for meta tags that usually contain publication dates, such as citation_date, pubdate, or DC.date. If found, then directly retrieves and normalizes the value (same format). If not, it searches for <time> tags in the HTML, especially those with a datetime attribute. This ensures that structured HTML elements are prioritized.

If to find both meta tags and time tags fail, the program then uses regular expressions (regex) as a fallback. It defines several regex patterns to match various date formats, like DD Month YYYY, YYYY-MM-DD, or MM/DD/YYYY. This step is designed to brute force through the full page text, looking for anything that resembles a date. Although this process is less precise, it ensures a broader coverage across different website structures.

This brute force matching is especially important for sources that not follow a standard structure. Once a match is found, the program passes the date string into a normalization function that tries multiple format conversions to standardize it as YYYY-

MM-DD. If all else fails, it attempts to at least extract the year as a fallback. This adaptive, multi-layered strategy allows the system to extract publication dates from a wide range of source types and layouts.

Table 3.3. Date Patterns

```
  date_patterns = [
                                        r'\b([A-Za-
z]+,\s*\d{1,2}\s+(Jan|Feb|Mar|Apr|Mei|Jun|Jul|Agu|
Sep|Okt|Nov|Des)\s+\d{4}(?:\s+\d{2}:\d{2})?)',
                    r'\b(\d{1,2}\s+(Jan|Feb|Mar|Apr|Mei|
Jun|Jul|Agu|Sep|Okt|Nov|Des)\s+\d{4})',
                    r'\b(\d{1,2}\s+(January|February|Mar
ch|April|May|June|July|August|September|October|No
vember|December)\s+\d{4})',
                    r'\b((January|February|March|April|M
ay|June|July|August|September|October|November|Dec
ember)\s+\d{1,2},\s*\d{4})',
                    r'\b(\d{1,2}\s+(Jan|Feb|Mar|Apr|May|
Jun|Jul|Aug|Sep|Oct|Nov|Dec)\s+\d{4})',
                    r'\b((Jan|Feb|Mar|Apr|May|Jun|Jul|Au
g|Sep|Oct|Nov|Dec)\s+\d{1,2},\s*\d{4})',
                   r'\b(\d{4}-\d{2}-\d{2})',
                   r'\b(\d{1,2}/\d{1,2}/\d{4})',
                   r'\b(\d{1,2}-\d{1,2}-\d{4})'
          ]
```

4. Other Extraction

Same with other functions, the program uses a structured and layered approach to extract additional citation details such as publisher, journal title, DOI, volume, issue, and page numbers. It first looks for specific meta tags, like citation_publisher, citation_journal_title, citation_doi, and so on. These tags are commonly used in academic pages and provide reliable values.

If meta tags are missing, some fields like DOI are searched in the full text using regular expressions. For example, the DOI is matched with the pattern doi:\s*(10.\d+/...), allowing the program to extract it even when it's embedded in plain text. This shows the fallback brute force technique, where the program scans all visible text to catch missing structured data. For page numbers, it combines citation_firstpage and citation_lastpage to generate a full page range like 123–130. This consistent and adaptive extraction ensures that even if one method fails, other strategies can still retrieve the needed information.

D. Citation Formatter

After retrieving and storing citation data in the CitationData class, then is formatting this data into proper citation styles. Each implementing a standard format such as APA, MLA, or Chicago. These formatter classes inherit from a base class CitationFormatter, which defines a common interface for formatting.

Example is the APAFormatter class defines how to format author names into "Last, F. M." style, handles publication dates in parentheses, and places the journal title, volume, issue, and pages in the correct APA layout. It also prioritizes the use of DOI when available, followed by a fallback to the article's URL.

On the other class, the ChicagoFormatter uses a more narrative style, often including the title in quotation marks, placing the date in parentheses, and appending the access date and URL at the end for online resources. The formatter also decides when to use "et al." for more than three authors.

Each formatter includes a helper method such as _join_authors, which ensures that authors are combined using the correct punctuation and conjunctions based on the citation style. This modular approach makes it easy to support additional formats in the future by creating new formatter classes.

So this formatting component translates raw citation data into professional, style-compliant references that can be directly used in scholarly writing or bibliography generation.

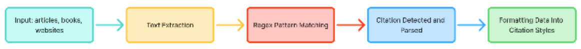
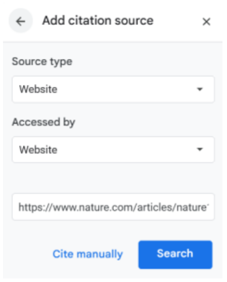

Input: articles, books, websites
Text Extraction
Regex Pattern Matching
Citation Detected and Parsed
Formatting Data Into Citation Styles

Figure 3.1. Program Workflow
Source: Personal Collection

IV. Testing and Analysis

The testing results show that the algorithm works well in extracting information from a URL and converting it into the selected citation style. One example test case used the URL: https://www.nature.com/articles/nature12373.



Add citation source

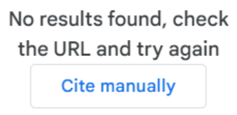
Source type: Website

Accessed by: Website

https://www.nature.com/articles/nature

Figure 4.1. Add Citation Source by Google Docs
Source: Personal Collection

When trying the automatic citation feature in Google Docs, it could not process the link correctly. It first asks the user to choose the type of source, and even after pasting the link, it often returns an error saying "no result found" and suggests checking the URL and try again.

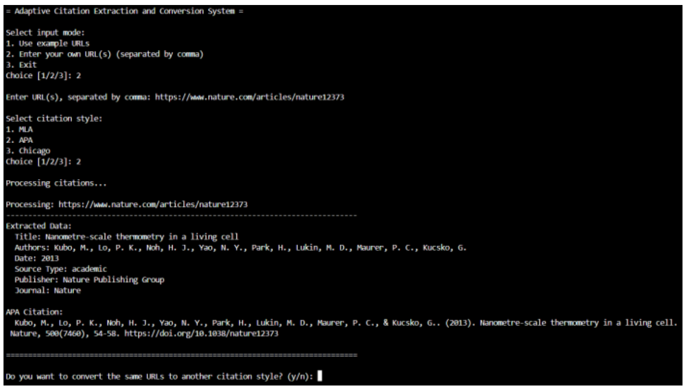


No results found, check the URL and try again

Figure 4.2. Result Citation by Google Docs
Source: Personal Collection

In contrast, this program automatically detects the type of source from the link and generates the correct citation without requiring any manual input. When the program starts, the user is prompted to choose an option, either to enter an example link, provide a custom URL (or multiple URLs), or exit the program.

In the example case, the program processes a single link and returns the citation in APA style. The output can be seen in the image below.



```
= Adaptive Citation Extraction and Conversion System =

Select input mode:
1. Use example URLs
2. Enter your own URL(s) (separated by comma)
3. Exit
Choice [1/2/3]: 2

Enter URL(s), separated by comma: https://www.nature.com/articles/nature12373

Select citation style:
1. MLA
2. APA
3. Chicago
Choice [1/2/3]: 2

Processing citations...

Processing: https://www.nature.com/articles/nature12373
--------------------------------------------------------------------------------
Extracted Data:
  Title: Nanometre-scale thermometry in a living cell
  Authors: Kubo, M., Lo, P. K., Noh, H. J., Yao, N. Y., Park, H., Lukin, M. D., Maurer, P. C., Kucsko, G.
  Date: 2013
  Source Type: academic
  Publisher: Nature Publishing Group
  Journal: Nature

APA Citation:
  Kubo, M., Lo, P. K., Noh, H. J., Yao, N. Y., Park, H., Lukin, M. D., Maurer, P. C., & Kucsko, G.. (2013). Nanometre-scale thermometry in a living cell.
 Nature, 500(7460), 54-58. https://doi.org/10.1038/nature12373

================================================================================

Do you want to convert the same URLs to another citation style? (y/n):
```

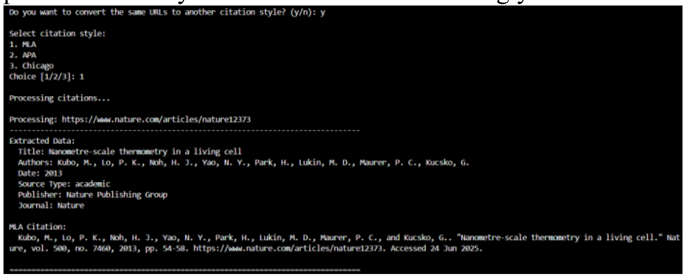
Figure 4.3. Result Citation from Program
Source: Personal Collection

One of the main strengths of this system is its effectiveness in adaptive extraction. Unlike conventional citation tools that require users to manually select the type of source (e.g., website, or journal article), this program automatically analyzes the content of the given URL and determines the appropriate source type. This enables the system to handle a wide range of inputs, such as academic papers, news articles, and blogs, without additional user intervention.

From the results, it is clear that this approach is effective in extracting and generating citations automatically. The system leverages pattern matching with regular expressions to search for relevant metadata such as dates, DOIs, publishers, and journal titles. This process resembles a brute force method, where multiple regex patterns are tried sequentially to find the correct match for each field. By systematically attempting all possible patterns, the system increases its flexibility and adaptability, ensuring successful extraction across various source structures and formats.

Another key advantage is the flexibility of regex matching itself. Different websites often format metadata in unique ways, using different HTML tag names or content structures. By preparing multiple regular expression patterns for each type of data, the system ensures broad compatibility. This approach allows it to adapt to inconsistencies in HTML structures, such as varying tag names, missing attributes, or different metadata conventions, making it more robust and reliable.

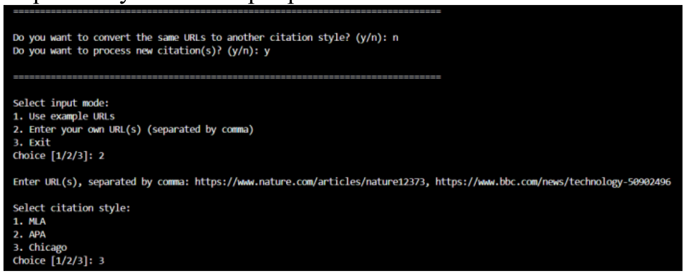
The program supports automatic conversion to multiple citation styles too. This is made possible through the separation between data extraction and citation formatting, which is managed by classes like APAFormatter and ChicagoFormatter. This modular design simplifies the process of generating output in different citation formats. Users can easily switch styles without modifying the extraction logic, and the system will produce correctly formatted citations accordingly.



```
Do you want to convert the same URLs to another citation style? (y/n): y

Select citation style:
1. MLA
2. APA
3. Chicago
Choice [1/2/3]: 1

Processing citations...

Processing: https://www.nature.com/articles/nature12373
--------------------------------------------------------------------------------
Extracted Data:
  Title: Nanometre-scale thermometry in a living cell
  Authors: Kubo, M., Lo, P. K., Noh, H. J., Yao, N. Y., Park, H., Lukin, M. D., Maurer, P. C., Kucsko, G.
  Date: 2013
  Source Type: academic
  Publisher: Nature Publishing Group
  Journal: Nature

MLA Citation:
  Kubo, M., Lo, P. K., Noh, H. J., Yao, N. Y., Park, H., Lukin, M. D., Maurer, P. C., and Kucsko, G.. "Nanometre-scale thermometry in a living cell." Nat
ure, vol. 500, no. 7460, 2013, pp. 54-58. https://www.nature.com/articles/nature12373. Accessed 24 Jun 2025.

================================================================================
```

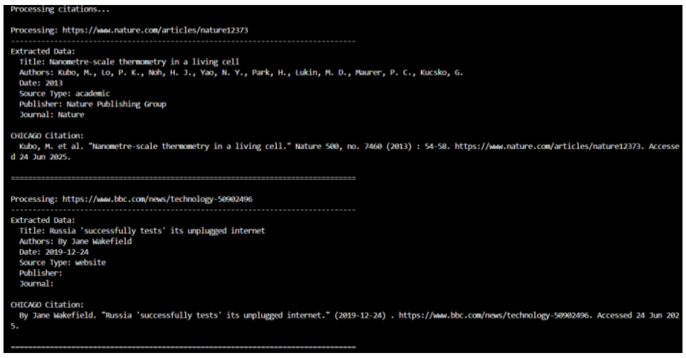
Figure 4.4. Automatic Conversion Citation from Program
Source: Personal Collection

This program also supports multiple URLs, making it efficient when there are many links that need to be converted into references. However, it is less efficient in terms of processing time, as it uses a brute-force approach that sequentially tries multiple patterns to find a match.



```
================================================================================

Do you want to convert the same URLs to another citation style? (y/n): n
Do you want to process new citation(s)? (y/n): y

================================================================================

Select input mode:
1. Use example URLs
2. Enter your own URL(s) (separated by comma)
3. Exit
Choice [1/2/3]: 2

Enter URL(s), separated by comma: https://www.nature.com/articles/nature12373, https://www.bbc.com/news/technology-50902496

Select citation style:
1. MLA
2. APA
3. Chicago
Choice [1/2/3]: 3
```

Figure 4.5. Result Citation from Program
Source: Personal Collection



```
Processing citations...

Processing: https://www.nature.com/articles/nature12373
--------------------------------------------------------------------------------
Extracted Data:
  Title: Nanometre-scale thermometry in a living cell
  Authors: Kubo, M., Lo, P. K., Noh, H. J., Yao, N. Y., Park, H., Lukin, M. D., Maurer, P. C., Kucsko, G.
  Date: 2013
  Source Type: academic
  Publisher: Nature Publishing Group
  Journal: Nature

CHICAGO Citation:
  Kubo, M. et al. "Nanometre-scale thermometry in a living cell." Nature 500, no. 7460 (2013) : 54-58. https://www.nature.com/articles/nature12373. Accesse
d 24 Jun 2025.

================================================================================

Processing: https://www.bbc.com/news/technology-50902496
--------------------------------------------------------------------------------
Extracted Data:
  Title: Russia 'successfully tests' its unplugged internet
  Authors: By Jane Wakefield
  Date: 2019-12-24
  Source Type: website
  Publisher:
  Journal:

CHICAGO Citation:
  By Jane Wakefield. "Russia 'successfully tests' its unplugged internet." (2019-12-24) . https://www.bbc.com/news/technology-50902496. Accessed 24 Jun 202
5.

================================================================================
```

Figure 4.6. Result Multiple Citation from Program
Source: Personal Collection

V. Conclusion

In conclusion, the implementation of pattern matching using regular expressions proves to be highly effective for identifying and generating citations directly from URLs. This approach ensures that all relevant text on a webpage is thoroughly scanned, either through structured HTML tags or flexible regex patterns, allowing for broad coverage and accurate data extraction across various types of online sources.

By systematically checking multiple predefined patterns, the system increases its chances of successfully retrieving citation-related information. However, one limitation is that not all possible pattern variations can be anticipated, as different websites may structure their content in unique and inconsistent ways. While this brute force strategy is effective in many cases, it can lead to higher computational costs and longer processing times.

Therefore, future development should focus on optimizing the extraction process by search more efficient and intelligent pattern matching strategies. Despite this, the program demonstrates promising functionality, especially with its ability to automatically detect source types and support multiple URL inputs, which significantly improves user convenience and citation accuracy.

VI. Appendix

Github Link: https://github.com/mineraleee/CitationExtraction
Video Link at Youtube: https://youtu.be/EbEMGmCrPn4

VII. Acknowledgment

All praise and gratitude are directed to God Almighty, for His blessings and guidance that have enabled me to complete this paper titled *"Adaptive Citation Extraction with Regex and String Matching for Multiple Source Types and Auto-Conversion"* on time. This paper was prepared as part of the assignment for the IF2211 Algorithm Strategy course.

I would like to express sincere appreciation to Dr. Ir. Rinaldi, M.T., for his dedicated guidance and for generously sharing his knowledge throughout the course. The insights provided have been invaluable in completing this paper.

I am especially thankful to my parents for their continuous support, both emotionally and financially, throughout this journey. Their encouragement has been a constant source of motivation. I also want to extend my heartfelt thanks to my friends in the Split Bill group for the collaboration, encouragement, and shared ideas that greatly enriched this project.

I recognize that this paper is not without flaws. I sincerely apologize for any errors or omissions and hope that this work can still provide meaningful insights and contribute, even in a small way, to the development of knowledge in this field.

References

[1] Lehigh University Libraries, "Citation and Plagiarism," [Online]. Available: https://libraryguides.lehigh.edu/citation. [Accessed: June 22, 2025].

[2] Munir, Rinaldi, "String Matching," [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf. [Accessed: June 22, 2025].

[3] Munir, Rinaldi, "String Matching dengan Regular Expression," [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/24-String-Matching-dengan-Regex-(2025).pdf. [Accessed: June 23, 2025].

[4] Munir, Rinaldi, "Algoritma Brute Force," [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/02-Algoritma-Brute-Force-(2025)-Bag1.pdf. [Accessed: June 23, 2025].

Statement

I hereby declare that the paper I wrote is my own writing, not an adaptation or translation of someone else's paper, and not plagiarized.

Bandung, 24 June 2025

[illegible]

Heleni Gratia M Tampubolon
(13523107)