STRING MATCHING (Using Regular Expression)

Abhay.M
B.Tech
School of Engineering
Hyderabad, India
2111CS020007@mallareddyuniversity.ac.in

Abhigynan.B
B.Tech
School of Engineering
Hyderabad, India
2111CS020008@mallareddyuniversity.ac.in

Abhilash Reddy.M
B.Tech
School of Engineering
Hyderabad, India
2111CS020009@mallareddyuniversity.ac.in

Abhinay.M
B.Tech
School of Engineering
Hyderabad, India
2111CS020010@mallareddyuniversity.ac.in

Abhinaya.C
B.Tech
School of Engineering
Hyderabad, India
2111CS020011@mallareddyuniversity.ac.in

Abhinaya.G
B.Tech
School of Engineering
Hyderabad, India
2111CS020012@mallareddyuniversity.ac.in

Guide: *Professor kalyani*
School of Engineering,
Mallareddy University

Abstract: The Regex Tool project is a web application that allows users to input a text string and a regular expression, and then checks for and displays any matching patterns. The tool is built using Python and the Django web framework. The project starts with the installation of Django and the setup of a new Django project. A new Django app is then created to handle the functionality of the Regex Tool. The app includes a view function that receives input from the user via a form and uses Python's re module to search for matches in the input string using the provided regular expression. The results are then displayed to the user on the same page. The front-end of the Regex Tool is built using HTML, CSS, and JavaScript, and uses Django's built-in template system to render the HTML templates. The user interface includes two input fields for entering the text string and the regular expression, as well as a submit button to initiate the search. The results of the search are displayed in a table, with each row representing a matching pattern and the columns showing the location of the match within the input string Overall, the Regex Tool project is a useful tool for anyone who needs to search for and analyze patterns in text data. It demonstrates the power of regular expressions and how they can be used in conjunction with Python to create a practical and user-friendly web application

Keywords:- Neural network, Convolutional neural network,
Character Recognition, Regular Expression

I. INTRODUCTION

Regular expressions, often abbreviated as "regex" are a powerful tool for working with text. They are essentially a set of rules and patterns that can be used to match and manipulate strings of text. Regular expressions are used in a wide range of applications, including search engines, text editors, and programming languages. They can be used to validate user input, search for specific patterns within a text document, and perform various other text manipulation tasks. Some of the most common uses of regular expressions include:

•Matching strings of text: Regular expressions can be used to search for specific patterns or sequences of characters within a string of text. For example, you could use a regular expression to find all instances of the word "cat" within a document.

•Validating user input: Regular expressions can be used to ensure that user input follows a specific format or pattern. For example, you could use a regular expression to ensure that a user's email address is properly formatted.

•Replacing text: Regular expressions can be used to replace specific patterns or sequences of characters within a string of text. For example, you could use a regular expression to replace all instances of the word "cat" with "dog" within a document. Regular expressions can be complex and take some time to master, but they are a powerful tool for anyone who works with text on a regular basis In real time, this tool can be used to quickly and easily search for specific patterns in a large body of text. For example, suppose you have a document with hundreds of email addresses and you need to extract only the email addresses that end with ".edu". Here's how you could use the regular expression tool to accomplish this task: Copy and paste the entire document into the text input box of the tool. In the regular expression input box, enter the pattern "(\w+@\w+.edu)". Click the "Submit" or "Search" button to initiate the search. The tool will scan the text and highlight all occurrences of email addresses that end with ".edu" in the output box. 2 You can then copy and paste the highlighted email addresses into a separate document or file for further processing. This is just one example of how regular expressions can be used to quickly and efficiently search for specific patterns in text. The tool can be used for a wide range of applications, such as extract -racting phone numbers, identifying URLs, validating input in forms, and much more.

II. PROBLEM STATEMENT

Design and implement a program that utilizes regular expressions to match and extract specific patterns from text data. The program should take a user-defined regular expression pattern and a text input as input and provide the corresponding matched results as output.

Features to include:

Regex Pattern Input: The program should allow the user to enter a regular expression pattern to match against the input text.

Text Input: The program should accept text input from the user, which will be processed and matched against the provided regular expression pattern.

Matching and Extraction: The program should use the expression pattern to match against the input text and extract the corresponding matched results. It should identify all occurrences of the pattern in the text and provide them as output.

Output Display: The program should display the matched results, indicating the portions of the text that match the specified pattern. Additionally, it should provide any captured groups within the pattern, if applicable.

regular

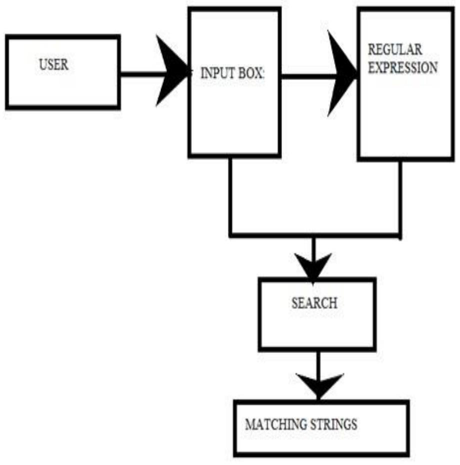


Fig: UML diagram for Regex matcher

III. LITERATURE REVIEW

The use of regular expressions (regex) has become an integral part of many software applications and programming languages. A regex matcher project involves developing an efficient algorithm or tool that can match patterns specified by regular expressions against input text. This literature review aims to provide an overview of relevant research and advancements in regex matching techniques.

The literature review highlights various regex matching algorithms and optimization techniques, ranging from traditional approaches like NFA and backtracking to more advanced methods like DFA and NFA simulation. Understanding the strengths and weaknesses of different algorithms can guide the development of an efficient and robust regex matcher project. Additionally, leveraging existing tools and libraries can provide insights into best practices and real-world implementations of regex matching.

IV. REQUIRED TOOLS

- Python
- Django
- Re-module

V. METHODOLOGY

Deterministic Finite Automata (DFA): This algorithm involves building a finite automata that reads a string character by character and decides whether the string matches a given pattern. The DFA is constructed by taking the regular expression and converting it into a state machine that can recognize strings matching the pattern. The DFA algorithm is useful for simple patterns, but it can become quite complex for more complex patterns.

Non-Deterministic Finite Automata (NFA): This algorithm works similar to the DFA algorithm, but it allows for multiple transitions from a single state on a single input character. This makes the NFA more flexible than the DFA and can handle more complex patterns. The NFA is converted to a DFA to speed up the pattern matching process.

Thompson's Construction: This algorithm involves building an NFA by recursively breaking down the regular expression into sub-expressions and combining them using operators such as concatenation, alternation, and closure. The resulting NFA can then be used to match strings against the regular expression.

Backtracking: This algorithm works by trying all possible paths through a regular expression until a match is found. It is often used by regex engines to handle complex patterns that cannot be handled by the other algorithms.

Recursive Descent Parsing: This algorithm involves recursively breaking down a regular expression into sub-expressions and then recursively matching the string against those subexpressions. This algorithm is often used by programming languages to parse regular expressions and build a parse tree.

VI. EXPERIMENT RESULTS

Fig:

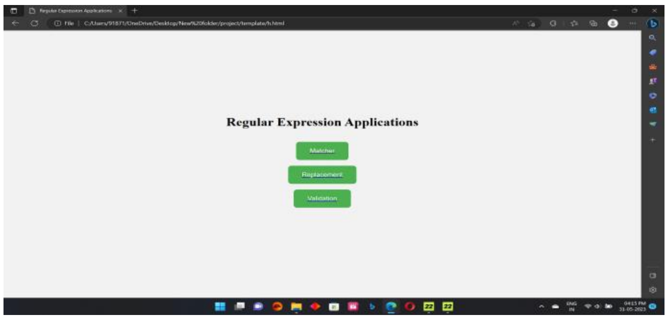
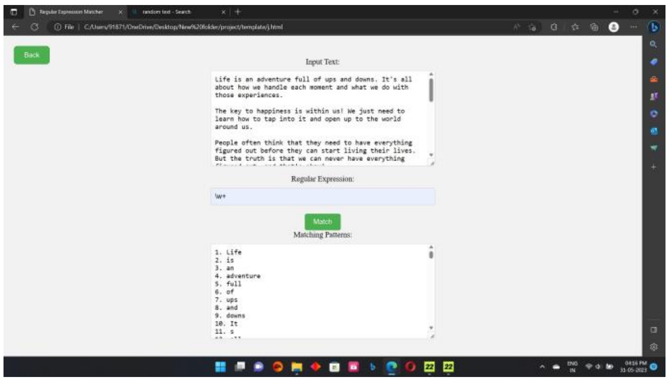


Fig:


Input Text:

```
Life is an adventure full of ups and downs. It's all
about how we handle each moment and what we do with
those experiences.

The key to happiness is within us! We just need to
learn how to tap into it and open up to the world
around us.

People often think that they need to have everything
figured out before they can start living their lives.
But the truth is that we can never have everything
```

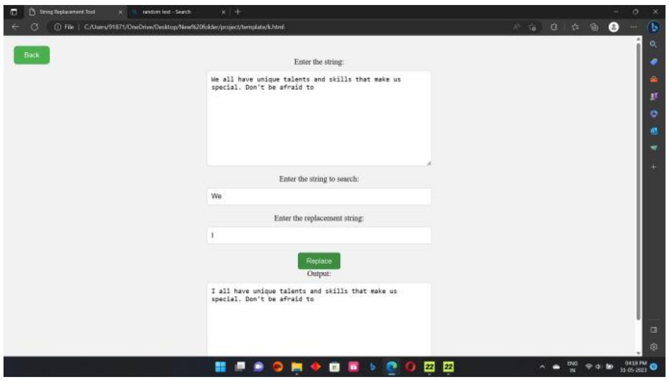
Regular Expression:

```
\w+
```

Matching Patterns:

```
1. Life
2. is
3. an
4. adventure
5. full
6. of
7. ups
8. and
9. downs
10. It
11. s
```

Fig:


Enter the string:

```
We all have unique talents and skills that make us
special. Don't be afraid to
```

Enter the string to search:

```
We
```

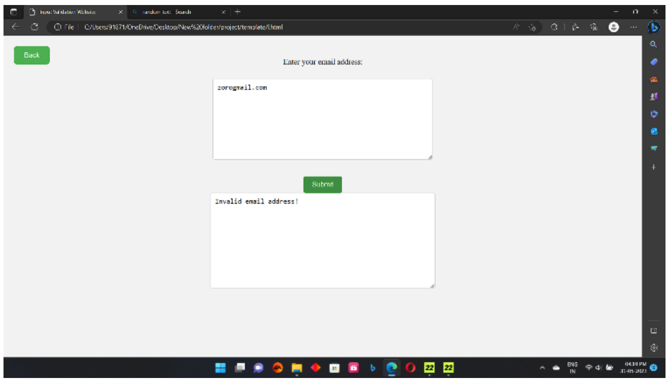
Enter the replacement string:

```
I
```

Output:

```
I all have unique talents and skills that make us
special. Don't be afraid to
```

Fig:


Enter your email address:

```
zorogmail.com
```

```
Invalid email address!
```

VII. MERITS OF PROPOSED SYSTEM

The proposed system for a regex matcher project offers several merits and benefits. Here are some of them:

Accuracy: The system aims to improve the accuracy of regex matching by carefully designing and testing patterns against a diverse set of sample data. Through iterative refinement and optimization, it strives to achieve precise matching results, minimizing false positives and false negatives.

Flexibility: Regular expressions provide a powerful and flexible tool for pattern matching. The proposed system harnesses this flexibility by allowing users

to define and customize patterns according to their specific needs. It enables the matching of complex patterns and supports various modifiers, quantifiers, and metacharacters.
Efficiency: The system emphasizes performance optimization techniques to ensure efficient matching of regular expressions. By applying optimizations such as precompilation, minimizing backtracking, and utilizing efficient quantifiers and boundary conditions, the system aims to deliver fast and responsive matching even for large input strings or complex patterns.
Versatility: Regular expressions can be utilized in a wide range of applications, including text parsing, data validation, search functionality, and more. The proposed system can be integrated into different software systems or applications to provide regex matching capabilities, enhancing their functionality and usefulness.
Documentation and Maintainability: The system promotes good documentation practices by documenting the regex patterns, optimizations applied, and any known limitations. This documentation helps future developers understand the implementation and facilitates maintenance, allowing for easier troubleshooting, bug fixes, or future enhancements.
Extensibility: The proposed system can be extended to accommodate additional features or requirements. As the project evolves, new patterns can be added, existing patterns can be modified, and optimizations can be further refined. This extensibility allows the system to adapt to changing needs and ensures its long-term viability.
Scalability: The system is designed to handle varying scales of input data and patterns. Whether dealing with small strings or large documents, the proposed system aims to provide efficient and accurate matching. By optimizing performance and considering edge cases, it strives to maintain scalability even as the dataset or pattern complexity increases.
Cost-Effectiveness: Regular expressions are a cost-effective solution for pattern matching tasks, as they are widely available and supported by numerous programming languages and libraries. The proposed system leverages these existing resources, reducing development costs and time-to-market.

VIII. ARCHITECTURE DIAGRAM

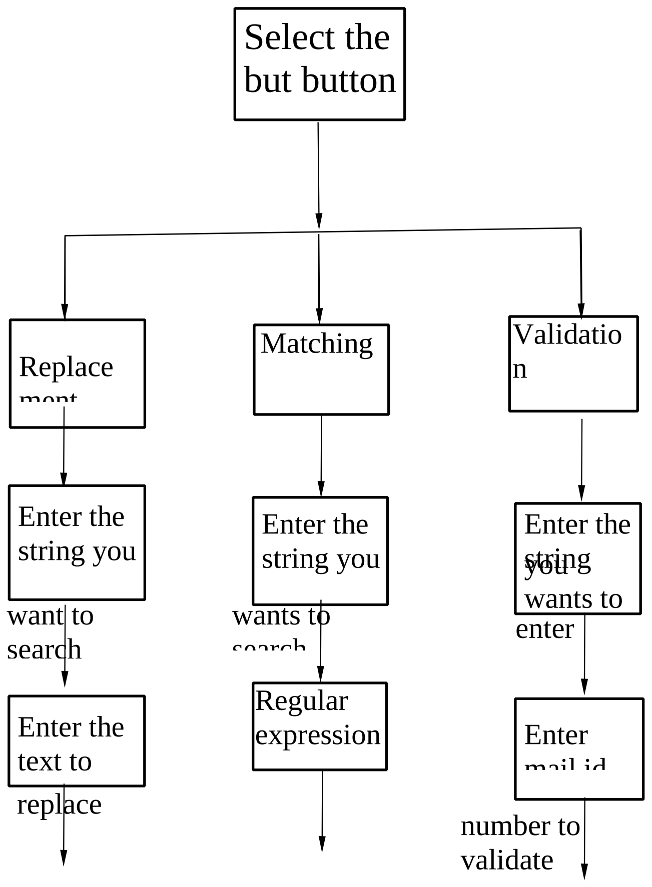

Select the but button
Replace ment
Matching
Validatio n
Enter the string you want to search
Enter the string you wants to search
Enter the string you wants to enter
Enter the text to replace
Regular expression
Enter mail id number to validate

Fig: Architecture

In conclusion, regular expressions provide a concise and flexible way to handle complex pattern matching tasks. They require a certain level of knowledge and skill to use effectively, but the effort is well worth it for anyone working with text data on a regular basis. With regular expressions, you can quickly and easily search, extract, and manipulate text data, making it an invaluable tool for data analysis, web development, and many other fields.

X. Future Enhancement:

Regular expressions have been around for several decades and have already undergone numerous enhancements and improvements. However, there is always room for further enhancements, and here are some potential areas for improvement:
1.Performance optimization: Regular expressions can be very powerful, but they can also be slow to

execute when working with large data sets. Future enhancements could focus on optimizing performance, possibly by using more advanced algorithms and data structures.

2.Natural language processing: While regular expressions are excellent at working with structured data, they struggle with natural language processing (NLP) tasks such as sentiment analysis or semantic parsing. Future enhancements could involve incorporating NLP capabilities into regular expressions to enable more advanced text analysis.

3.Improved syntax: The syntax for regular expressions can be difficult to learn and use effectively. Future enhancements could focus on making the syntax more intuitive and userfriendly, possibly by introducing more natural language-like expressions.

4.Cross-platform standardization: Different programming languages and platforms may have their own implementations of regular expressions, leading to inconsistencies in behavior and syntax. Future enhancements could involve developing a cross-platform standard for regular expressions to ensure consistent behavior across all platforms.

Overall, regular expressions are already a powerful tool, but there is always room for improvement. These enhancements could make regular expressions even more versatile and effective, enabling them to handle a wider range of text processing task.

ACKNOWLEDGMENT

We are so thankful to Prof. kalyani of department Artificial intelligence & Machine learning (AIML) for her guidance and support. We consider ourself to be extremely privileged to have been her students. We benefited enormously from her excellence as a professor and as a researcher. We are very grateful to her for being patient and for all her time that she spent in discussing about the project to guide us. We are Immensely grateful for her helpful discussion, support, and encouragement throughout the Project. We gave our best under her guidance. Finally, we consider ourselves to be extremely fortunate to have had the to opportunity to do project under the guidance of prof. Kalyani. We also express our heartfelt gratitude to Dr. Thayyaba Khatoon (Head of the Department), for giving all of us such a wonderful opportunity to explore ourselves and the outside world to work on the real-life scenarios where the Artificial Intelligence is being used nowadays.