SEO Keyword Tracker and Analyzer

Group 5

Rohit Bhaskar Uday(ID: 884451915)

Viditi Vartak(ID: 885146886)

Tushar Yadav(ID: 885174284)

Rohit Chowdary Yadla(ID:885146852)

Huan Ying(ID:885193656)

Shiyun Zhou (ID:884425810)

Department of Computer Science, Cal State Fullerton

CPSC 535

Dr. Syed Hassan Shah

April 8, 2024

TABLE OF CONTENTS

1. INTRODUCTION **2**
2. METHODOLOGY **3**
2.1 Integration of Python Scripts 3
2.2 Algorithm Implementation 4
2.2.1 Knuth-Morris-Pratt (KMP) Algorithm 4
2.2.2 Naive String Matching Algorithm 5
2.2.3 Rabin-Karp Algorithm 5
2.2.4 Suffix Array Algorithm 6
2.2.5 Suffix Tree Algorithm 7
2.3 Tool Development Process 7
2.4 Testing & Refinement 8
3. FEATURES **9**
3.1 Keyword Tracking 9
3.2 Keyword Density Analysis 10
3.3 Recommendations 10
3.4 User Interface 10
3.5 Additional Features 11
4. CHALLENGES & SOLUTIONS **11**
5. INSIGHTS & APPLICATIONS **13**
6. CONCLUSION **14**
REFERENCES **14**

1. INTRODUCTION

The goal of the "SEO Keyword Tracker and Analyzer" project is to provide SEO professionals, webmasters, and content creators with a revolutionary tool that will simplify the process of monitoring and optimizing web content, thereby improving its visibility and performance in search engines. The core of the project is the integration of a series of advanced string matching and analysis algorithms: Rabin-Karp, Suffix Tree, Suffix Array, Naive String Matching and Knuth-Morris-Pratt (KMP) algorithm. Each of these algorithms brings unique advantages, from efficient pattern searching to rapid identification of keyword occurrences through complex data structures, together forming the powerful analytical engine of this tool.

Combining these Python scripts not only enhances the tool's performance but also ensures the reliability and accuracy of keyword tracking and analysis. The Rabin-Karp algorithm efficiently scans large amounts of text for keyword patterns through its hash-based search method, reducing the time required for analysis. Suffix trees and suffix arrays provide compact representations of substrings in text data, making the search and analysis of keyword density and distribution fast. At the same time, the excellent performance of naive string matching and the KMP algorithms in direct pattern matching ensures that keyword occurrences are accurately located. Leveraging the strengths of these algorithms, the project offers a versatile platform that adapts to the diverse and dynamic nature of SEO challenges. It provides SEO professionals with actionable insights, from keyword density analysis to optimization recommendations, thereby supporting improved search engine rankings and web visibility.

2. METHODOLOGY

2.1 Integration of Python Scripts

The integration of Python scripts into the SEO Tracker and Analyzer Tool involves the careful selection of specialized Python libraries, incorporation, and coordination of various scripts to enable effective data processing, analysis, and presentation.

At the core of the integration process is the utilization of specialized Python libraries and modules for specific tasks within the tool. For instance, the ‘streamlit’ library serves as the backbone for creating the tool's interactive user interface, providing functionalities for data input, visualization, and user interaction. Additionally, libraries such as ‘requests’, ‘BeautifulSoup’, and ‘validators’ are used for web scraping, data retrieval, and URL validation. These libraries help the SEO Tracker and Analyzer Tool capability to gather relevant data from web sources in dynamic times.

Moreover, the integration extends to custom-built Python scripts designed to execute specific tasks essential for SEO tracking and analysis. These scripts include functions for text preprocessing, string-matching algorithms, data manipulation, and statistical analysis. Each script is carefully crafted to address particular aspects of the SEO analysis process, contributing to the overall functionality and effectiveness of the tool.

Furthermore, the integration process involves ensuring compatibility and interoperability among the integrated scripts. This includes handling data transfer, format conversion, and error handling to ensure smooth communication and coordination between different components of the tool. Additionally, version control practices are implemented to manage updates, revisions, and dependencies among the integrated scripts, ensuring consistency and reliability in the tool's operation.

2.2 Algorithm Implementation

The implementation of the SEO Tracker and Analyzer Tool involved five key algorithms: the Knuth-Morris-Pratt (KMP) algorithm, naive string matching algorithm, Rabin-Karp algorithm, suffix array algorithm, and suffix tree algorithm.

2.2.1 Knuth-Morris-Pratt (KMP) Algorithm

The KMP algorithm is effective in pattern-matching tasks. It offers a linear-time solution for searching for occurrences of a pattern within a text. The algorithm's key advantage lies in its ability to avoid unnecessary character comparisons by utilizing information about previously

matched characters. By preprocessing the pattern to construct a partial match table, also known as the failure function, the KMP algorithm can efficiently skip over unmatched characters in the text, leading to improved performance in pattern-matching tasks.

The ‘compute_prefix_function(pattern)’ function calculates and returns the prefix function pi for a given pattern and initializes an array pi of length m, where m is the length of the pattern. It iterates through the pattern and updates the values of pi based on the longest proper prefix of the pattern, which is also a suffix of the pattern. The function then returns the computed prefix function pi.

The ‘kmp_search(text, pattern)‘ function performs the KMP search algorithm to find occurrences of the pattern in the given text.It first computes the prefix function pi for the pattern. Then, it iterates through the text, updating the state j, which is the number of characters matched based on the prefix function pi. When a match is found, which means j equals the length of the pattern m, it records the occurrence and continues the search. Finally, the function returns the total number of occurrences found.

2.2.2 Naive String Matching Algorithm

The naive string-matching algorithm involves iteratively comparing the pattern with all substrings of the text to identify occurrences of the pattern. While the naive approach may not offer the same level of efficiency as more advanced algorithms, it serves as a baseline method for pattern-matching tasks and is particularly useful for small-scale analyses or scenarios where performance is not a critical concern.

In the code implementation ‘naive_string_matching(text, pattern)’ function takes two input parameters: text, which is the text to be searched, and pattern, the pattern to be matched. It initializes an empty list of occurrences to store the starting indices of matches. Then, It iterates through the text using a loop with a range from 0 to n - m + 1, where n is the length of the text and m is the length of the pattern. This ensures that the pattern can fit within the remaining portion of the text. Within the loop, it checks if the substring of length m starting at index i in the

text matches the pattern. If a match is found, it adds the index i to the occurrences list. Finally, it returns the total number of occurrences found, which is the length of the occurrences list.

2.2.3 Rabin-Karp Algorithm

The Rabin-Karp algorithm offers a hashing of the pattern and substrings of the text to identify potential matches quickly. The algorithm's efficiency stems from its ability to compare hash values instead of individual characters, reducing the number of comparisons required. While the Rabin-Karp algorithm may exhibit lower worst-case performance compared to other algorithms, it excels in scenarios where hash collisions are rare, offering a viable solution for pattern-matching tasks.

The code implementation for Rabin-Karp involved four functions. The '__init__'function initializes the RabinKarp object with the provided text and pattern. It calculates the lengths of the text and pattern, initializes hash values, and sets the base and prime values for the hashing operation.

The 'calculate_hash_value'function calculates the hash value of a given string of specified length.It iterates through the string, computing the hash value using the base and prime numbers for the modulo operation.

Then 'recalculate_hash_value' recalculates the hash value when sliding the window of the text. It updates the hash value based on the old hash value, the character leaving the window (old_char), and the character entering the window (new_char).

The 'search_pattern' function performs the pattern search using the Rabin-Karp algorithm. It iterates through the text, comparing hash values of the pattern and substrings of the text. If a potential match is found based on hash values, it performs a character-wise comparison to confirm the match. It keeps track of the count of occurrences and returns it.

2.2.4 Suffix Array Algorithm

The suffix array algorithm provides an efficient data structure for storing all suffixes of a text in sorted order. By constructing a suffix array, the algorithm enables rapid substring searches and pattern-matching operations within the text. Its simplicity and effectiveness make it a popular choice for various text-processing tasks, including substring searches, longest common substring detection, and pattern matching.

The ‘build_suffix_array’ function constructs the suffix array for the given text. It appends a special character '$' to the end of the text to mark the end of the text. Then, it creates a list of tuples (text[i:], i) representing suffixes of the text along with their starting indices. The list of suffixes is then sorted lexicographically. Finally, it extracts the indices of the sorted suffixes to build the suffix array and returns it.

The ‘count_pattern_occurrences’ function counts the occurrences of a pattern within the given text using the constructed suffix array. It first builds the suffix array of the text using the build_suffix_array function. It defines two inner functions, binary_search_first and binary_search_last to perform binary search to find the first and last occurrences of the pattern in the suffix array, respectively. The binary_search_first function searches for the first occurrence of the pattern in the suffix array. The binary_search_last function searches for the last occurrence of the pattern in the suffix array. The function returns the count of occurrences of the pattern, calculated as the difference between the indices of the first and last occurrences plus one.

2.2.5 Suffix Tree Algorithm

The suffix tree algorithm extends the capabilities of the suffix array by constructing a compact trie-like data structure representing all suffixes of a text. This data structure enables efficient substring searches, pattern matching, and other text processing operations. The suffix tree's compact representation and fast query times make it well-suited for handling large text corpora and complex pattern-matching tasks.

In the implementation of the suffix tree algorithm, we import the Suffix Tree Module STree class from the suffix_trees module, which is presumably used to construct and manipulate suffix trees.

Then, the ‘count_pattern_occurrences’ function takes the text to be searched and the pattern to be matched as parameters. It first creates a suffix tree st from the given text using the STree class constructor. It then searches for all occurrences of the pattern within the suffix tree using the find_all method of the suffix tree. The function returns the total number of occurrences found by taking the length of the list occurrences.

2.3 Tool Development Process

To guarantee the "SEO Keyword Tracker and Analyzer" tool's efficacy and usability for SEO experts, a methodical development procedure was employed. Below is a summary of the essential steps:

- Analysis of Requirements: To understand the unique demands and pain points of SEO specialists in keyword tracking and analysis, the team conducted in-depth interviews and surveys with them. Determining desired functionality, data sources to be integrated, and performance standards were all part of this process.
- Design Phase: The group created a modular tool architecture based on the requirements analysis. This required specifying the elements for keyword analysis (using techniques such as Suffix Tree, Rabin-Karp, and others), data visualization, reporting, and data retrieval (web scraping, API integration). The user interface and workflow were visualized using wireframes and mockups.
- Implementation: To construct an interactive user interface, Python scripts for algorithmic features were integrated, and libraries such as "streamlit" were utilized. To support the main features of the program, special scripts were created for data preparation, statistical analysis, and web scraping.
- Integration Testing: To guarantee smooth operation and communication between all components, extensive integration testing was carried out. This involved evaluating the retrieval of data from online sources, precise keyword analysis through the use of algorithms, and appropriate result display.
- User Acceptance Testing (UAT): For UAT, a beta version of the program was made available to a certain set of SEO experts. Their comments and recommendations were

taken into consideration to enhance the tool's functionality, reduce any performance snags, and increase usability.

- Record-keeping: Throughout the development process, thorough documentation encompassing installation instructions, usage recommendations, algorithm descriptions, and troubleshooting advice was kept up to date. The users' and developers' complete guide was provided by this documentation.

2.4 Testing & Refinement

Ensuring the accuracy, functionality, and user happiness of the tool was the main goal of the testing and improvement phase. These particular actions were taken:

- Validating keyword tracking accuracy, algorithmic findings against existing datasets, and the appropriate operation of data visualization tools were all part of the functional testing process. Test cases were created to address a range of situations and edge circumstances.
- Performance Testing: To assess the tool's scalability and speed, performance testing was carried out. Large datasets were used for testing in order to gauge memory utilization, response times, and the tool's capacity to manage several user requests at once.
- Usability Testing: To get input on the tool's user interface, navigation, and general user experience, usability testing sessions were held with SEO specialists. The process and user interface were iteratively improved by analyzing user feedback.
- Bug Fixing and Optimization: Performance problems and defects were quickly identified, prioritized, and fixed. The main goals of optimization were to increase algorithm performance, speed up data processing, and maximize resource use.
- Feature Enhancements: To improve the tool's functioning and give SEO experts more information, new features were introduced based on user input and testing outcomes. This involved connecting with more data sources, enhancing data visualization capabilities, and introducing new analytical metrics.

Through a thorough process of testing and refining that was customized to meet the unique objectives of the project, the "SEO Keyword Tracker and Analyzer" tool was optimized to

provide SEO experts with accurate keyword monitoring, effective analysis, and an intuitive user interface.

TestCases

TestCase_id	Test_Case	TestCase_Description	Expected Output	Status
TC_01	GUI Layout	User should be able to open the application to ensure the GUI layout is visually appealing and components are well-organized.	GUI is launched and all the components are well organized.	PASS
TC_02	Labels	Verify that all buttons and input fields are appropriately labeled.	All the fields and labels are labeled correctly.	PASS
TC_03	URL	User should be able to enter the URL in the input field.	URL is entered in the field by the user.	PASS
TC_04	Algorithm select	User should be able to select any algorithm of choice.	The algorithm is selected.	PASS
TC_05	Select multiple algorithms.	User should be able to select more than one algorithms.	Multiple algorithms are selected.	PASS
TC_06	Analyze button	User should be able to click on analyze button.	Analyze button is working.	PASS
TC_07	Naïve string matching algorithm	User should be able to run using Naïve string matching algorithm.	The application runs successfully.	PASS
TC_08	Kmp algorithm	User should be able to run using Kmp algorithm.	The application runs successfully.	PASS

TC_09	Rabin karp algorithm	User should be able to run using rabin karp algorithm.	The application runs successfully.	PASS
TC_10	Suffix array	User should be able to run using suffix array.	The application runs successfully.	PASS
TC_11	Suffix tree	User should be able to run using Suffix tree.	The application runs successfully.	PASS
TC_12	Multiple algorithms	User should be able to run using more than one algorithms.	The application runs successfully.	PASS
TC_13	All words table	All word tables should be generated.	All word tables are generated.	PASS
TC_14	Word cloud	Word cloud should be generated.	Word cloud is generated.	PASS
TC_15	Top 10 words bar graph	A bar graph with top 10 words should be generated.	A bar graph with top 10 words is generated.	PASS
TC_16	Processing time chart	Processing time chart should be generated.	Processing time chart is generated.	PASS
TC_17	Scatter plot	Scatter plot should be generated.	Scatter plot should be generated.	PASS
TC_18	Word length distribution	Word length distribution graph should be generated.	Word length distribution graph is generated.	PASS

3. FEATURES

3.1 Keyword Tracking

The keyword tracking feature allows users to monitor and track keyword performance on a specified URL. Users can enter the URL they want to analyze directly into the tool's input field. The tool then runs the selected algorithms (Rabin-Karp, Suffix Array) to track the specified keywords on the given URL. This feature provides valuable insights into keyword occurrences, positions, and trends on the analyzed webpage.

3.2 Keyword Density Analysis

The keyword density analysis feature calculates the frequency of keywords within the content of the analyzed URL. Users can select the "Words" option to perform keyword density analysis. The tool generates a report highlighting the density of each keyword within the content, helping users understand keyword prominence and optimize content for better search engine visibility.

3.3 Recommendations

The recommendations feature provides actionable insights based on keyword tracking and density analysis results. Users can visualize the top keywords by selecting the "Top 10 Words" option. Additionally, the "Word Cloud" feature generates a visual representation of keyword frequency, aiding in identifying important keywords. Based on these analyses, the tool offers recommendations for optimizing content, improving keyword usage, and enhancing SEO strategies.

3.4 User Interface

The user interface (UI) of the "SEO Tracker and Analyzer Tool" includes intuitive elements for seamless navigation and data visualization. Users can interact with the tool by entering the URL to analyze and selecting algorithms (Rabin-Karp, Suffix Array) for keyword tracking. The UI also provides options for displaying processing times, visualizing data with a scatter plot, viewing word length distribution, and accessing the all words table. These features ensure a user-friendly experience and facilitate efficient analysis of SEO data.

3.5 Additional Features

In addition to the core functionalities, the "SEO Tracker and Analyzer Tool" offers advanced features for in-depth analysis and insights:

Scatter Plot: The scatter plot feature visualizes keyword occurrences or densities against specific parameters, providing a graphical representation of keyword trends and correlations.

Word Length Distribution: This feature analyzes the distribution of word lengths within the content of the analyzed URL. It helps users understand the structure and readability of the content, optimizing for user experience and SEO.

All Words Table: Users can access a comprehensive table containing all words extracted from the analyzed URL. The table provides detailed information about word frequencies, occurrences, and positions, aiding in thorough keyword analysis and content optimization.

These features collectively enhance the tool's functionality, providing SEO professionals with valuable insights, actionable recommendations, and tools for optimizing keyword performance and content strategy.

4. CHALLENGES & SOLUTIONS

Challenge 1: Integrating Different String Matching Algorithms

Integrating various string matching algorithms like Rabin-Karp, Suffix Tree, Suffix Array, Naive String Matching, and the KMP algorithm was initially challenging. Each algorithm has its complexities and was designed for different contexts.

Solution: The development team created a unified interface to interact with all algorithms seamlessly. By abstracting the core functionality into a consistent API, the team enabled the tool to utilize any of these algorithms interchangeably based on the user's selection. This approach facilitated the easy incorporation of algorithms into the tool's processing flow, ensuring that adding or modifying algorithms in the future would be straightforward.

Challenge 2: Optimizing Performance for Large Datasets

Handling large datasets efficiently, especially when scraping and analyzing data from extensive web pages, posed a significant performance bottleneck.

Solution: To enhance performance, the team employed multi-threading for web scraping tasks and optimized our string-matching algorithms for speed. For instance, optimizations were applied to the Suffix Array and Tree constructions and utilized efficient hash calculations in the Rabin-Karp algorithm. Additionally, caching mechanisms were implemented to store previously analyzed results, reducing the need for repeated calculations.

Challenge 3: User Interface Design

Creating a user-friendly and intuitive interface for a tool that handles complex operations behind the scenes was another challenge.

Solution: The choice of Streamlit for its simplicity and effectiveness in creating interactive web applications was made. The GUI was designed to be straightforward, allowing users to input URLs, select algorithms, and view results in various formats, including tables, word clouds, and charts. This choice significantly reduced development time and provided an easy-to-navigate interface for users.

Challenge 4: Accuracy in Keyword Relevance and Stopword Removal

Ensuring the relevance of keywords and effectively removing stopwords to improve analysis accuracy was challenging, given the diversity of web content.

Solution: A comprehensive list of stopwords was curated, and the provision for custom stopword additions tailored to specific domains was made. Algorithms were also implemented to analyze the context in which keywords appeared, ensuring that the tool only considered relevant keywords. This approach significantly improved the tool's ability to identify and analyze keywords accurately.

Challenge 5: Error Handling and URL Validation

Handling errors gracefully to prevent the app from crashing and to provide useful feedback to the user. Also, ensuring that users input valid URLs for analysis.

Solution: We implemented try-except blocks in the code to catch exceptions and used Streamlit's ‘st.error’ method to display user-friendly error messages. We integrated a URL validation function using the ‘validators’ library in Python. This function checks the user's input and provides feedback if the URL is invalid.

5. INSIGHTS & APPLICATIONS

Insights: The development process of the SEO Keyword Tracker and Analyzer tool brought to light the critical role of algorithm selection in handling various types of text data. Specifically, it was found that the Knuth-Morris-Pratt (KMP) algorithm shows a remarkable efficiency in processing structured and repetitive texts, whereas Suffix Trees are more adept at dealing with unstructured data featuring diverse keyword occurrences. Furthermore, the tool's analysis capabilities unveiled trending SEO strategies, emphasizing the growing importance of long-tail keywords and the necessity for meticulous content optimization based on keyword density and relevance. These insights are instrumental for enhancing website visibility and search engine rankings.

Application: The SEO Keyword Tracker and Analyzer tool finds its utility in several key areas of digital marketing and SEO strategy formulation. Content marketers stand to gain substantial benefits from utilizing the tool to identify and capitalize on high-value keywords within their target niche, gain insights into competitors' keyword strategies, and tailor their content to better fit search engine algorithms. Additionally, the tool proves to be a cornerstone in conducting comprehensive SEO audits for websites. By offering detailed analyses of a website's keyword performance and pinpointing areas needing improvement, it enables businesses to effectively benchmark their performance against competitors and make informed decisions aimed at bolstering their online presence and search engine rankings. Therefore, this tool can help anyone in digital marketing, SEO, or content creation who's looking to make their content work harder online.

6. CONCLUSION

The SEO Keyword Tracker and Analyzer tool bridges the gap between complex algorithmic theory and practical SEO needs, offering a robust solution for keyword tracking and analysis. Throughout its development, emphasis was placed on user-friendly design and the strategic application of various string matching algorithms, highlighting the tool's potential to democratize sophisticated SEO analysis.

Looking ahead, the tool's functionality could be significantly enhanced by integrating predictive analytics through machine learning, expanding language support for global applicability, and introducing real-time data tracking for dynamic SEO adjustments. Additionally, API integrations could streamline its incorporation into broader digital marketing ecosystems.

As the digital landscape continues to evolve, the tool's adaptability will be key to its ongoing relevance. Future updates focusing on machine learning, language diversity, and seamless integration with existing marketing tools promise to enhance its utility, making it an indispensable asset for digital marketers, SEO specialists, and content creators aiming to optimize online content effectively.

REFERENCES

1. Google. "Search Engine Optimization (SEO) Starter Guide | Google Search Central." *Google Developers*, 2022, developers.google.com/search/docs/beginner/seo-starter-guide.

2. Moz. "The Beginner's Guide to SEO." *Moz*, 2019, moz.com/beginners-guide-to-seo.

3. Hardwick, Joshua. "12 Quick SEO Tips to Increase Organic Traffic." *SEO Blog by Ahrefs*, 1 Sept. 2020, ahrefs.com/blog/seo-tips/.

4. Patel, Neil. "SEO Made Simple: A Step-By-Step Guide for 2019." *Neil Patel*, 2019, neilpatel.com/what-is-seo/.

5. Goodwin, Danny. "What Is SEO / Search Engine Optimization?" *Search Engine Land*, 2023, searchengineland.com/guide/what-is-seo.

6. "The Backlinko SEO Blog by Brian Dean." *Backlinko*, backlinko.com/blog.

7. SEO Strategy Guide: https://blog.hubspot.com/marketing/seo-strategy-guide