****

**Department of Computer Science Engineering**

**INFORMATION RETRIEVAL AND SEMANTIC WEB**

**Project Report**

Submitted by:

**Arsh Gupta (19103068) B2**

**Siddhant Kushwaha (19103196) B6**

**Shivam Gupta (19103204) B6**

**Piyush Saxena (19103207) B6**

Submitted to:

**Dr. Neetu Sardana**

**String Similarity Search using Edit Distance and Soundex Algorithm**

**Abstract**

String similarity is a major inquiry that has been generally used for DNA sequencing, error tolerant, auto completion, and data cleaning which is required in database, data warehousing, and data mining. String similarity search is possible in various methodologies of procedures like Edit distance and Soundex algorithm. These strategies can be applied for long strings, which are not possible by the current methodologies on the grounds that the extent of the record constructed and an opportunity to manufacture such list. We apply distinctive string similitude strategies and check which procedure gives progressively suitable qualities. Our similarity measures incorporate Edit distance, and Soundex algorithm.

**INTRODUCTION**

String similarity search is a major question that has been utilized for DNA sequencing, error tolerant inquiry auto fulfillment, and information cleaning which is required in database, information distribution center, and information mining. In separation edit(s,t) ,between two string s and t, the string likeness seek is to discover each string in a string database D which is like a question string s with the end goal that edit(s,t)<=ĩ ,between two strings, s and t, the string closeness look is to discover each string t in a string database D which is comparable in an inquiry string s, to such an extent that edit(s,t) for a given edge ĩ

**THEORY**

**EDIT DISTANCE**

Given the edit distance, ed(s1,s2); between two strings, s1 and s2, is the minimum number of operations i.e., substitution, insertion, and deletion required to transform one string to another string[6]. Examples: Ant edit distance -1 Ani Bag edit distance -2 Uig Hit 3 substitution operations; Pan edit distance – 3

**ALGORITHM**

if(str[i]==str[j]) T[i][j] = T[i-1[j-1] //diagonal value else T[i][j] = min(T[i-1[j],T[i-1][j-1],T[i][j-1])//min of left, top, diagonal values

**ENHANCED EDIT DISTANCE**

The hypothesis is that a weighted edit distance model, with lower costs for keys that are closer together, will more often retrieve the desired search result when querying a search engine that allows for spelling correction. The hypothesis is based on the idea that one is more likely to accidentally press a key adjacent to the one intended rather than one far away from it. Additionally, it assumes that mechanical errors(typos) are common enough that the model benefits overall from capturing them more accurately at the expense of errors of ignorance, that is cases when the user does not know the correct spelling of the entity they are searching for. Those types of errors are not differentiated by the method even if they are present in the data.

**ALGORITHM**

To make the distance measurement the QWERTY-layout was first turned into a graph where the nodes are keys and there are edges between any two keys that are adjacent on the keyboard (adjacent meaning that the keys would touch if there was no spacing between the keys on the keyboard – making s and e adjacent but not t and h) After doing this, the distance between two keys was defined as the length of the shortest path between them in the graph, and the distance from any key to itself being the same as the distance to its neighbors. An initial distance matrix was constructed. The linear variant was constructed to have evenly distributed values from 0 to 2. On the keyboard, the shortest distance between any two keys is 1 (or 0, if you include the distance between a given key and itself), and the longest is 9,so the distance in number of keys was multiplied by 2/9 to get the desired distribution. The average distance in this matrix was calculated, and each value was then divided by this average. This normalization produces a matrix of weights where the expected value is 1 when choosing a cell at random.

**Deletion** **:** Originally, deletion was intended to be weighted according to whichever distance on the keyboard is shorter between the character to be deleted and the adjacent characters in the string. For example, removing the s from asmmunition to make ammunition would be weighted by the distance between a and s. However, doing that gives a benefit to any weighted model, because of the choosing of the shorter distance. Instead, deletion is weighted by the average of the distances to the adjacent characters in the string.

**Insertion :** Unchanged, weight 1. Missing a key is presumed not to be affected by their placement. It seems more like to result from mental error, rather than mechanical, because otherwise something would be pressed.

**Substitution :** Weighted according to the distance between the character that is removed and the character that is inserted. For example, swapping the s in buttsr for an e to make butter would be weighted by the distance between e and s.

**SOUNDEX ALGORITHM**

Soundex is a phonetic estimation for requesting names by sound, as enunciated in English. The goal is for homophones to be encoded to a comparative depiction with the objective that they can be facilitated not withstanding minor complexities in spelling. The figuring chiefly encodes consonants; a vowel won't be encoded except if it is the principal letter. SOUNDEX-TYPICAL

**ALGORITHM**

1. Hold the main letter of the word

2. Presently, Change every one of the events of the accompanying letters to '0' 'A','E','I','O','U','H','W','Y'

3. Change letters do digits as pursues: B,F,P,V - > 1 C,G,J,K,Q,S,X,Z - > 2 D,T - > 3 L - > 4 M,N - > 5 R - > 6

4. Evacuate all sets of successive digits.

5. Expel every one of the zeros from the subsequent string. 6. Include the subsequent string with trailing zeros and return the initial four positions, which will be of the structure

**IMPLEMENTATION**

**DESIGN**

**DESCRIPTION**

**o Frontend**

1. On the landing page, it consists of an elegant user interface and an interactive search bar where the user has to search for a word, And based on the searched word, top 5 closest words will be displayed.
2. It will display the top search results separately for each algorithm and a chart will also be displayed which will show the analysis of each algorithm and its performance.
3. Then there is a second page where the results based on the enhanced version of above algorithms and also their performance analysis will be displayed.

**o Backend**

1. we have implemented several versions of edit distance algorithm (unoptimized, optimized and enhanced) and Soundex algorithm.
2. In First version of edit distance, we have used a 2-d matrix,  
   its optimized version used 2 1-d arrays to boil down the space complexity to linear, to further optimize it, we used a single 1-d array.
3. And in the enhanced version, we first made a graph of the keyboard keys, adding an edge between each adjacent key and then we performed bfs algorithm to calculate the shortest distance between each pair of keys in the graph, then we normalized the values as we want the data to be distributed in the range 0 to 2.

This normalization produces a matrix of weights where the expected value is 1 when choosing a cell at random.

In the case of **deletion** of a character from Levenshtein algorithm, we will add the weighted values of the distance between the characters and same in the case of **substitution** while in the case of **insertion**, we add 1.

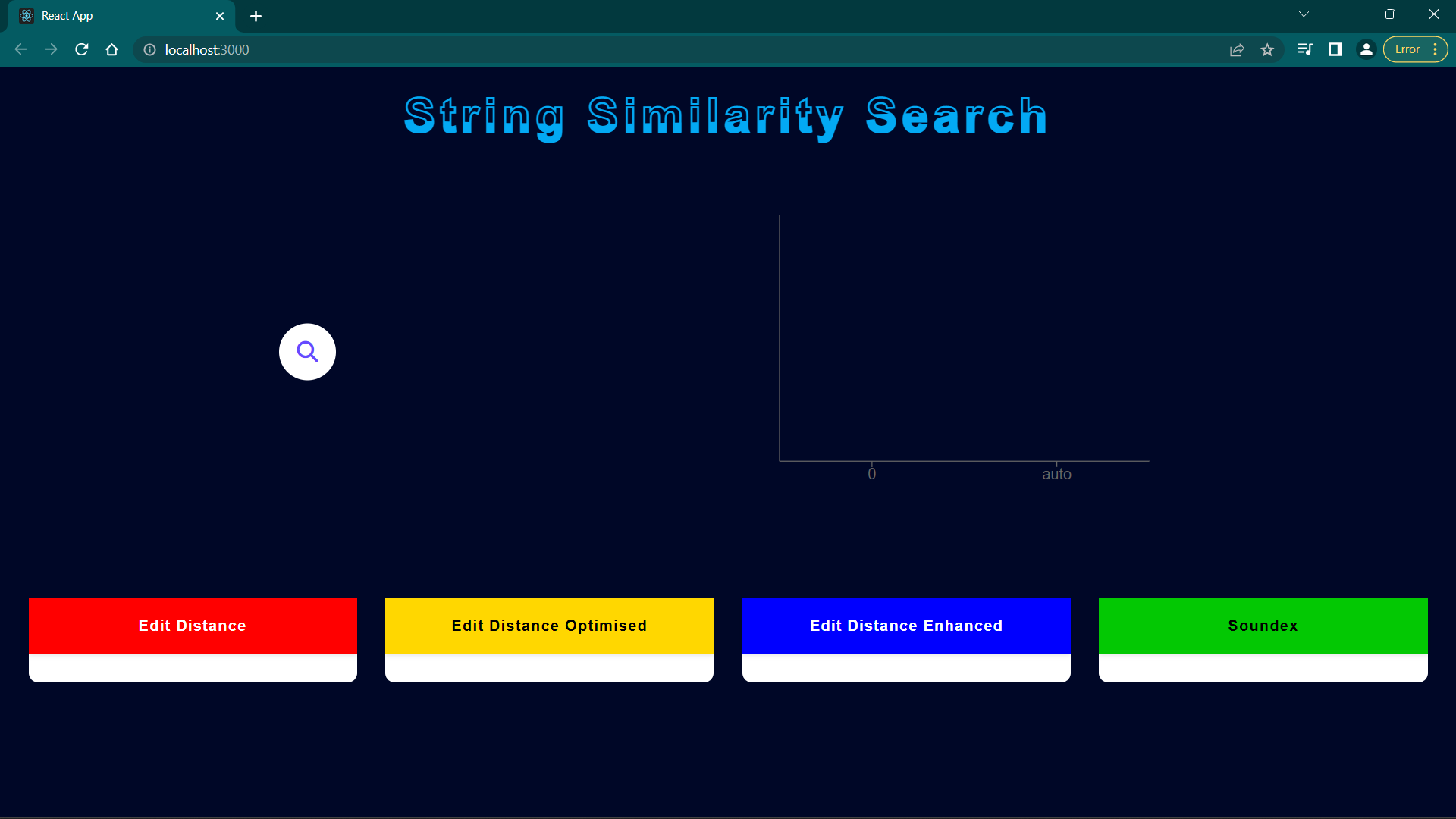
**Performance Parameters**

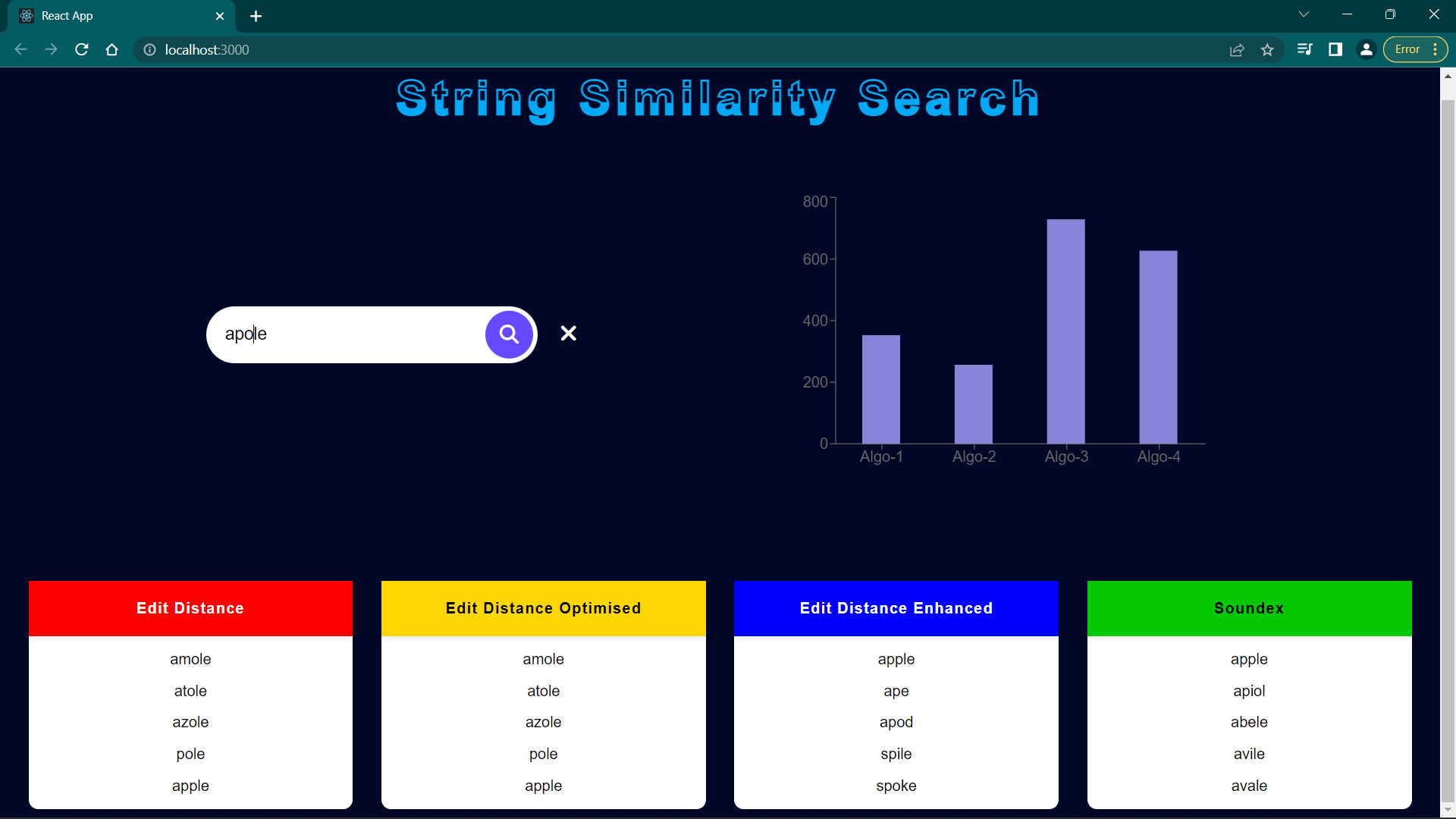
We have analyzed the performances of the various algos we have used in our project by measuring the time they are taking to give the desired results. We have used the performance.now() function which helps in getting the execution time of the code till now. We have taken the difference of the before and after times measured from the function.

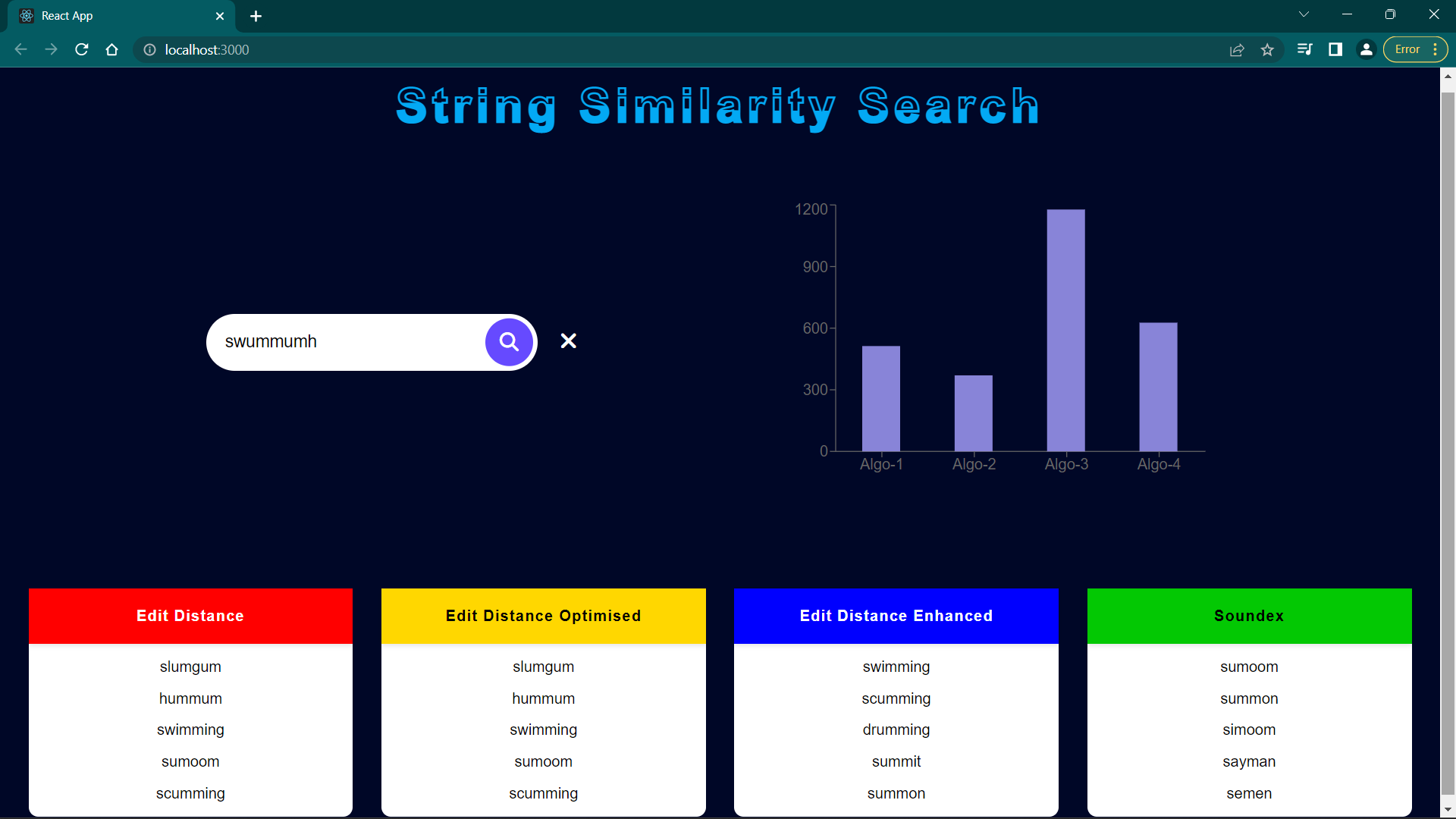
**RESULTS**

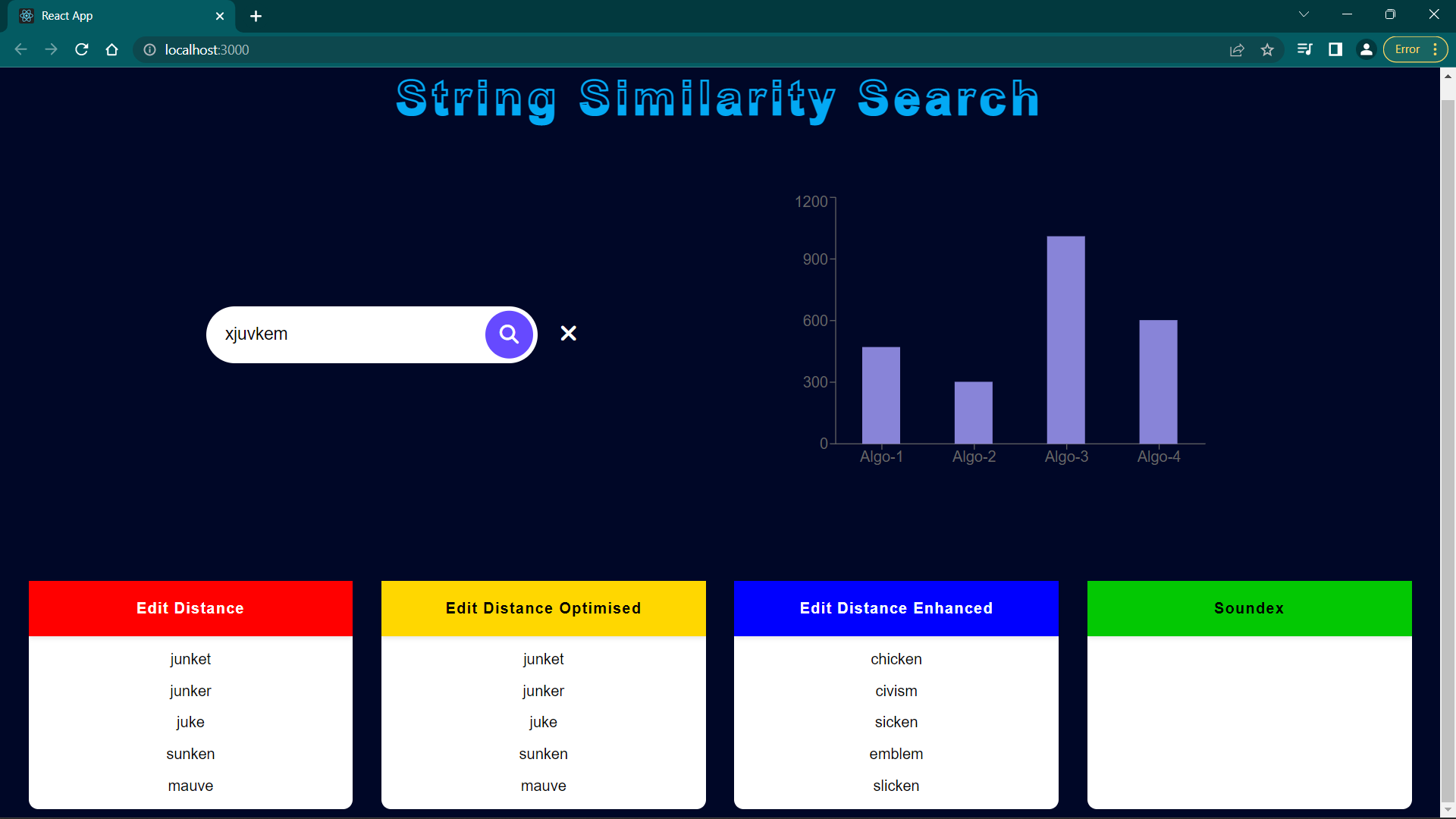
By using different string similarity techniques, we found how much strings are similar to each other so that those strings can be sent to data cleaning, data warehousing and data mining purposes.

**SCREENSHOTS**









**SOURCE CODE**

**TOOLS AND TECHNOLOGIES USED**

* **REACT.JS**
* **HTML5**
* **CSS3**
* **JAVASCRIPT**
* **VSCODE**
* **REACT CHARTS**
* **REDUX**

**DATASET USED**

* We have extracted the dataset from the following Github repository :-

<https://github.com/matthewreagan/WebstersEnglishDictionary>

* Our dataset is stored in JSON format

**CONCLUSION**

By using different string similarity approaches we decide whether given strings are similar or not and also, we check whether the strings are similar enough for data cleaning, data warehousing and data mining. These approaches are not only applicable for strings but they also differentiate various documents, files, etc.., The similar data values grouped together in the form clusters using k-means clustering algorithm

**REFERENCES**

<https://www.ijeat.org/wp-content/uploads/papers/v8i4/D6181048419.pdf>

<https://towardsdatascience.com/fuzzy-string-matching-algorithms-e0d483c2a9ea>

<http://www.diva-portal.org/smash/get/diva2:1116701/FULLTEXT01.pdf>

<https://www.kdnuggets.com/2020/10/optimizing-levenshtein-distance-measuring-text-similarity.html>

<https://www.scitepress.org/papers/2016/59263/59263.pdf>

<https://medium.com/@yash_agarwal2/soundex-and-levenshtein-distance-in-python-8b4b56542e9e>

<https://mrpowers.medium.com/fuzzy-matching-in-spark-with-soundex-and-levenshtein-distance-6749f5af8f28>