**Title: Plagiarism & Spam Checker**

**SOFTWARES USED**

1. **Visual Studio Code:** Visual Studio Code is a free source-code editor made by Microsoft for Windows, Linux and macOS.



1. **GitBash:** Git Bash is a source control management system for Windows. It allows users to type Git commands that make source code management easier through versioning and commit history.



1. **MinGW:** MinGW, formerly mingw32,is a free and open source software development environment to create Microsoft Windows applications.



**Introduction:**

Plagiarism & Spam Checker is a project designed to mainly check the “common content” between the Content stored and the content submitted by the user.  
It also lists out the Spam/Inappropriate words from the submitted work.

**Objective:**

Basically, the code would be programmed in such a way that it is able to analyse the whole content and tell the users how much content of theirs is plagiarised or is taken from someone else’s work or simply lacks originality.  
There will be a Local Database created for the working of this code. This Database would contain the content which would be used for checking the plagiarism for the content that the user would provide to us.

The user would be asked for the input of their content. When the user has given his content as an input, this is where programme comes into working. The programme would analyse the content and would tell us, in terms of percentage, that how much of the data is Plagiarised of the user.  
  
The code is also programmed in such a way that it is able to identify and print all the Spam or Inappropriate words that has been entered by the user.  
These words would also be stored in some local Database, for the code to identify it as an Inappropriate word.

**Basic classification of search algorithms:**

The various algorithms can be classified by the number of patterns each uses.

1.Single-pattern algorithms

• Naïve string-search algorithm

• Rabin–Karp algorithm

• Knuth–Morris–Pratt algorithm

• Boyer–Moore string-search algorithm

• Longest Common Substring algorithm

2. Algorithms using a finite set of patterns

• Aho–Corasick string matching algorithm (extension of Knuth-Morris-Pratt)

• Commentz-Walter algorithm (extension of Boyer-Moore)

• Set-BOM (extension of Backward Oracle Matching)

• Rabin–Karp string search algorithm.

**Algorithm Description:**

**Rabin Karp Algorithm Approach:**

A string search algorithm which compares a string's hash values, rather than the strings themselves. We are taking a help of Karp-Rabin Algorithm. It uses fingerprints to find occurrences of one string into another string. Karp-Rabin Algorithm reduces time of comparison of two sequences by assigning hash value to each string and word. Without hash value, it takes too much time for comparison like if there is a word W and input string is S then word is compared with every string and sub string in program and hence it consumes moretime.

Karp-Rabin has introduced concept of Hash value to avoid time complexity O(m2). It assigns hash value by calculating to both word and string/substring. So hash of substring (S) matches with hash value of W then only we can say exact comparison is done.

**Hash Values:**

A hash value is a numeric value of a fixed length that uniquely identifies data. The most wonderful character of Hash Values is that they are highly unique. No two data can theoretically have same Hash Value.

Karp-Rabin algorithm preferred category from left to right comparison. Function of hash must able to find has value efficiently. When first time name would be hashing with the same hash it save the data causing yields a value which will be compared to at data is index with the value.

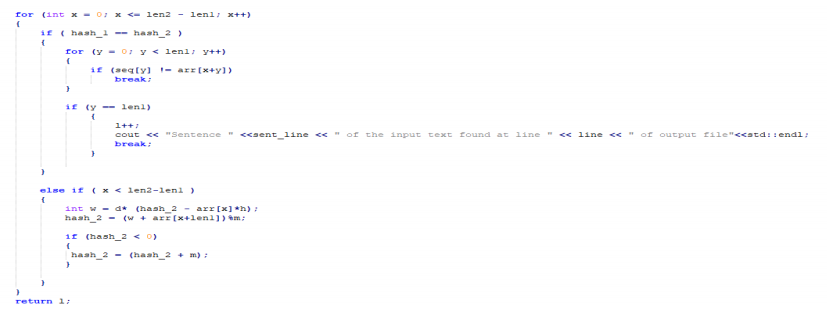
It can deal with multiple pattern matching that’s why people preferred this Karp-Rabin algorithm. Otherwise behavior of other algorithm is to perform basic pattern matching.

Its having O(nm) complexity. Where n is length of text and m is length of pattern. It is little bit slow also due to we have to check every single character from the text. The steps followed for algorithm development are as follows:

• We have taken String s and an input file and the patterns taken are each sentence separated by delimiter full stop in String s.

• For each sentence we checked if the sentence is matched with any of the lines of input files.

• At last we have kept two counters :- m for total number of sentences in the string s and n for total number of matched sentences of string s.



**ALOGORITHM USED FOR PLAGIARSM CHECKER:**

This algorithm is based on the concept of hashing, so if you are not familiar with string hashing.

Rabin Karp algorithm matches the hash value of the pattern with the hash value of current substring of text, and if the hash values match then only it starts matching individual characters. So Rabin Karp algorithm needs to calculate hash values for following strings.

1) Pattern itself.

2) All the substrings of the text of length m.

This algorithm was authored by Rabin and Karp in 1987.

**EXAMPLE:**

Problem: Given two strings - a pattern s and a text t, determine if the pattern appears in the text and if it does, enumerate all its occurrences in O(|s|+|t|) time.

Algorithm: Calculate the hash for the pattern s. Calculate hash values for all the prefixes of the text t. Now, we can compare a substring of length |s| with s in constant time using the calculated hashes. So, compare each substring of length |s| with the pattern. This will take a total of O(|t|) time. Hence the final complexity of the algorithm is O(|t|+|s|): O(|s|) is required for calculating the hash of the pattern and O(|t|) for comparing each substring of length |s| with the pattern.

**PSEUDO CODE:**

RABIN-KARP-MATCHER(T. P.d.q)

1. n = T.length
2. m = P.length
3. h = dm-1 mod q
4. p = 0
5. t0 = 0
6. for i = 1 to m // preprocessing
7. p = (dp Pill) mod q
8. to = (dro + mod q
9. For s =0 to n-m // matching
10. if p == ts
11. If P[1..m] ==T[s + 1..s + m]
12. print “Number occurs with shift” s
13. If s< n - m
14. Ts+1 = (d(ts -T[s+1]h) + T[s + m + 1]) mod q

**SAMPLE CODE:**

float rabinKarp(char seq[], char arr[], int line, int sent\_line)

{

//seq is the sentence of the input text and arr corresponds to lines of the output file.

int m = 121;

int d = 256;

int y;

int len1 = strlen(seq);

int hash\_1 = 0;

int len2 = strlen(arr);

int hash\_2 = 0;

int h = 1;

int l = 0;

for (int x = 0; x < len1 - 1; x++)

{

h = (h \* d) % m;

}

//For calculating the hash code value.

for (int x = 0; x < len1; x++)

{

hash\_1 = (d \* hash\_1 + seq[x]) % m;

hash\_2 = (d \* hash\_2 + arr[x]) % m;

}

for (int x = 0; x <= len2 - len1; x++)

{

if (hash\_1 == hash\_2)

{

for (y = 0; y < len1; y++)

{

if (seq[y] != arr[x + y])

break;

}

if (y == len1)

{

l++;

// cout << "Sentence " << sent\_line << " of the input text found at line " << line << " of output file" << std::endl;

break;

}

}

else if (x < len2 - len1)

{

int w = d \* (hash\_2 - arr[x] \* h);

hash\_2 = (w + arr[x + len1]) % m;

if (hash\_2 < 0)

{

hash\_2 = (hash\_2 + m);

}

}

}

return l;

}

**ALOGORITHM USED FOR SPAM CHECKER:**

Trie is an efficient information reTrieval data structure. Using Trie, search complexities can be brought to optimal limit (key length). If we store keys in binary search tree, a well balanced BST will need time proportional to **M \* log N**, where M is maximum string length and N is number of keys in tree. Using Trie, we can search the key in O(M) time.

**PSEUDO CODE:**

boolean check(String s)

{  
 for(every char in String s)

{

If(child node of current char is null)

{

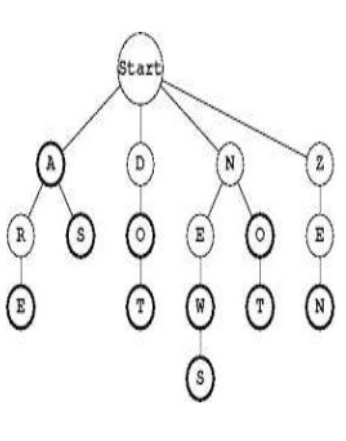
Return false;

}

}

Return true;

}



**SAMPLE CODE:**

bool search(string key)

{

TrieNode \*pCrawl = root;

for (int i = 0; i < key.length(); i++)

{

int index = key[i] - 'a';

if (!pCrawl->children[index])

return false;

pCrawl = pCrawl->children[index];

}

return (pCrawl != NULL && pCrawl->isEndOfWord);

}

} Spam;