Koneru Lakshmaiah Education Foundation

(Deemed to be University)

FRESHMAN ENGINEERING DEPARTMENT

A Project Based Lab Report

On

Anagram Search

SUBMITTED BY:

2400030325 - Siva

2400030386 – Chakradhar

2400030371 - PM Bhargav Sai

2400030417 - Tejesh Venkat Ready

UNDER THE GUIDANCE OF

B. Surendra Babu

Assistant Professor,CSE.

KL UNIVERSITY

Green fields, Vaddeswaram – 522 502

Guntur Dt., AP, India.

DEPARTMENT OF BASIC ENGINEERING SCIENCES-1

CERTIFICATE

This is to certify that the project based laboratory report entitled **ANAGRAM SEARCH** submitted by Mr.**PM BHARGAV SAI** bearing Regd. No. **2400030371** to the **Department of Basic Engineering Sciences-1, KL University** in partial fulfillment of the requirements for the completion of a project based Laboratory in "**Computational Thinking for Structural Design** "course in I B Tech II Semester, is a bonafide record of the work carried out by him/her under my supervision during the academic year 2023 – 2024.

PROJECT SUPERVISOR	**HEAD OF THE DEPARTMENT**
BELLAM SURENDRABABU	Dr. D. Haritha

ACKNOWLEDGEMENTS

It is great pleasure for me to express my gratitude to our honorable President **Sri. Koneru Satyanarayana**, for giving the opportunity and platform with facilities in accomplishing the project based laboratory report.

I express the sincere gratitude to our Director **Dr. A.Jagadeesh** for his administration towards our academic growth.

I express sincere gratitude to HOD-BES-1 **Dr. D.Haritha** for her leadership and constant motivation provided in successful completion of our academic semester. I record it as my privilege to deeply thank for providing us the efficient faculty and facilities to make our ideas into reality.

I express my sincere thanks to our project supervisor **BELLAM SURENDRABABU** for his novel association of ideas, encouragement, appreciation and intellectual zeal which motivated us to venture this project successfully.

Finally, it is pleased to acknowledge the indebtedness to all those who devoted themselves directly or indirectly to make this project report success.

Project Associates…

ABSTRACT

Anagram search is a computational problem that involves identifying all valid anagrams of a given input word or phrase from a set of potential candidate words. An anagram is a word or phrase formed by rearranging the letters of another, using all the original letters exactly once. The task of anagram searching has applications in areas such as natural language processing, cryptography, puzzle solving, and information retrieval. This paper explores efficient methods for performing anagram searches, including traditional sorting-based approaches, hash-based techniques, and advanced algorithms leveraging trie data structures and parallel processing. The study also evaluates the trade-offs between time complexity, memory usage, and accuracy, offering insights into how different algorithms perform under varying conditions, such as dictionary size and input word length. In addition, we present a practical implementation for large-scale anagram search systems, considering factors such as scalability and real-time query handling. The results demonstrate the efficacy of optimized search algorithms in quickly and accurately identifying anagrams, paving the way for more efficient language processing applications.

INDEX

INTRODUCTION

An anagram search system is a tool designed to find and display all anagrams of a given word from a set of potential candidate words (usually a dictionary). An anagram is a word formed by rearranging the letters of another word, using all the original letters exactly once. This system can be particularly useful in applications such as word games (Scrabble, Boggle), cryptography, or natural language processing tasks.

The main goal of this system is to efficiently search for anagrams within a given dictionary, optimizing both the time complexity and memory usage for large datasets. The system will allow users to input a word and retrieve a list of anagrams for that word from a dictionary**.**

Efficiency and Speed: C is known for its speed, which is crucial when dealing with large datasets or performing time-sensitive tasks like real-time anagram searching

Data Structures: C’s powerful support for arrays, linked lists, and hash tables enables the implementation of efficient algorithms for storing and searching anagrams.

AIM

The aim of developing the Anagram Search System in C is to create a software application that can efficiently search for anagrams of a given input word from a predefined dictionary of words. This involves:

1. **Efficient:** By implementing optimal algorithms, the system should quickly identify anagrams.
2. **Memory Management:** Proper handling of memory is essential to ensure the system can scale for large dictionaries.
3. **User Interaction**: Although C does not natively support graphical interfaces, the system will provide a command-line interface (CLI) that is simple and easy to interact with.

Advantages :-

1. **Efficiency and Speed:** C is known for its speed, which is crucial when dealing with large datasets or performing time-sensitive tasks like real-time anagram searching.
2. **Portability:** The system can run on different operating systems and hardware platforms with minimal changes to the code, which makes it suitable for various environments.
3. **Memory Management:** C allows low-level control of memory, ensuring that the system is optimized for handling large dictionaries with minimal memory consumption.
4. **Data Structures**: C’s powerful support for arrays, linked lists, and hash tables enables the implementation of efficient algorithms for storing and searching anagrams.
5. **Integration Potential:** The system can easily be integrated with other applications, such as word games or NLP tools, for further functionalities like scoring or text analysis.

Disadvantages of Implementing in C:

1. **Limited User Interface (UI) Capabilities**: C does not provide built-in support for GUI development, so the user interface will be text-based.
2. **Prone to Errors:** As a low-level language, C requires manual memory management, making it more error-prone, especially with pointers and memory allocation.
3. **Difficulty with Modularization:** While C supports modularization, its lack of object-oriented features can make the system more challenging to design and maintain compared to higher-level languages.
4. **Limited Libraries:** C does not have a large selection of ready-to-use libraries compared to modern languages, so much of the functionality (such as handling strings or hash tables) needs to be implemented from scratch.

System Architecture:

The Anagram Search System will use the following architecture:

1. **Input Interface:** A simple command-line interface (CLI) where the user inputs a word for which they want to find anagrams.
2. **Dictionary Data:** A predefined dictionary of words that will be stored in a file or an array. The system will load the dictionary into memory at runtime.
3. **Anagram Search Algorithm:** The core algorithm will be based on sorting the characters of the input word and comparing it with the sorted characters of words in the dictionary. Alternatively, hash-based techniques can also be employed.
4. **Output:** The system will output a list of words from the dictionary that are anagrams of the input word.

Future Enhancements:

1. **Multi-Threaded Search**: Implement multi-threading to allow concurrent searching when handling large datasets, thus improving performance on multi-core systems.
2. **Web Interface**: While C is not ideal for web applications, a wrapper or API can be created to expose the anagram search functionality via a web interface using another language like Python or JavaScript.
3. **Advanced Search Options:** Allow users to filter results by word length or word categories (e.g., nouns, verbs) for more specialized anagram searching.
4. **Real-Time Dictionary Update:** Integrate with online word databases or dictionaries to allow real-time updates of the word list without restarting the application.
5. **Machine Learning Integration:** Use machine learning models to suggest related words or categorize anagrams based on context.

System Requirements:

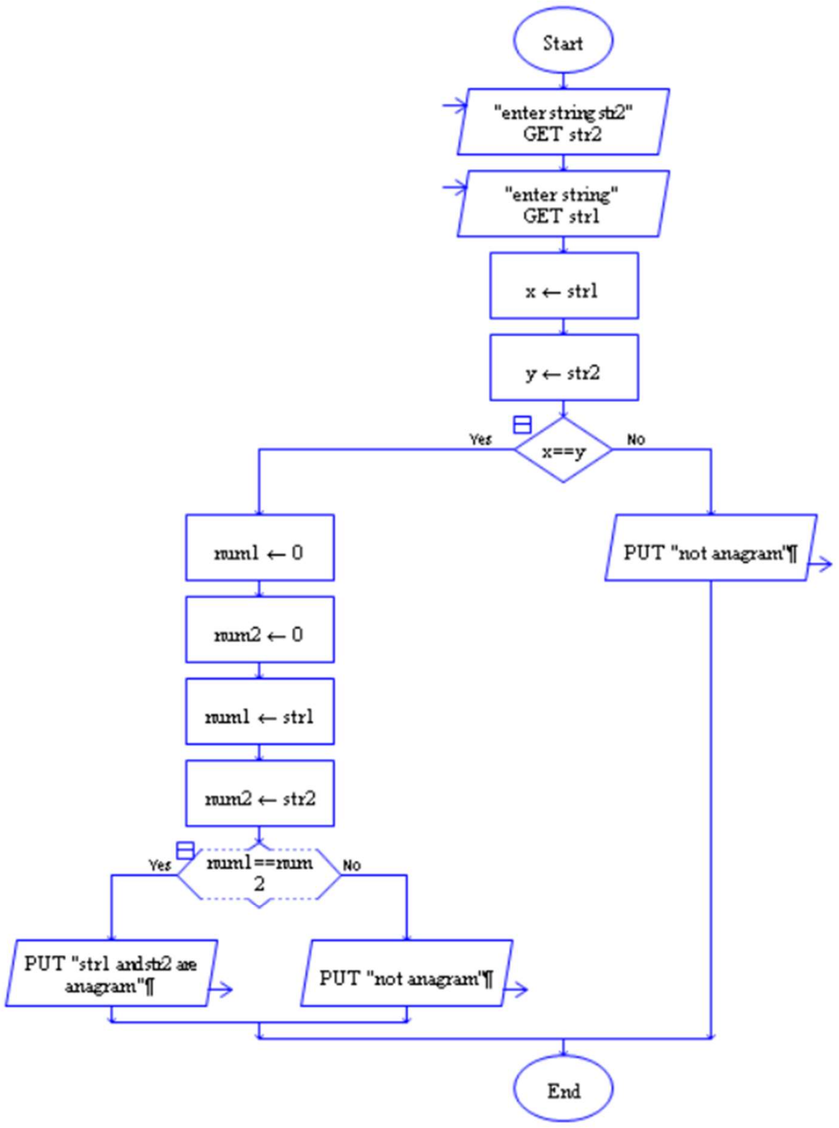
Software Requirements:

- Language: C Programming Language
- Operating System: Windows XP or later, Linux, or macOS
- Compiler: GCC, Turbo C, or any compatible C compiler

Hardware Requirements:

- RAM: At least 4 GB of RAM for smooth operation with smaller datasets; 8 GB or more for larger datasets.
- Processor: Intel Core i3, i5, or equivalent AMD processor for basic performance.
- Storage: 1 GB of free disk space for storing dictionary files and logs.

FLOW CHART


Start
"enter string str2"
GET str2
"enter string"
GET str1
x ← str1
y ← str2
x==y
Yes
No
num1 ← 0
num2 ← 0
num1 ← str1
num2 ← str2
num1==num
2
Yes
No
PUT "str1 and str2 are anagram"¶
PUT "not anagram"¶
PUT "not anagram"¶
End

ALGORITHM

1. Start: Begin the program.

2. Initialize Arrays: Create two integer arrays num1 and num2 of size 26 (for each letter of the alphabet) and initialize them to zero.

3. Read Input Strings: Read the two input strings str1 and str2 from the user.

4. Length Check: Compare the lengths of the two strings.

    - If the lengths are different, print "Strings are not anagrams" and exit.
    - If the lengths are the same, proceed to the next step.

5. Count Frequencies in str1:

    - Loop through each character in str1.
    - If the character is a lowercase letter (between 'a' and 'z'), increment the corresponding position in the num1 array.
    - If the character is an uppercase letter (between 'A' and 'Z'), increment the corresponding position in the num1 array.

6. Count Frequencies in str2:

    - Loop through each character in str2.
    - If the character is a lowercase letter (between 'a' and 'z'), increment the corresponding position in the num2 array.
    - If the character is an uppercase letter (between 'A' and 'Z'), increment the corresponding position in the num2 array.

7. Compare Arrays:

    - Loop through each position in the arrays num1 and num2.
    - If any position in num1 does not match the corresponding position in num2, print "Strings are not anagrams" and exit.

8. Print Result:

    - If all positions match, print "Strings are anagrams".

IMPLEMENTATION

```
/* Write a C program that takes two strings as input and checks whether
two strings are anagrams. */
#include <stdio.h>
#include <string.h>
int findanagram(char str1[], char str2[])
{
        int num1[26]={0},num2[26]={0},i=0;
        // check's length same or not
        if(strlen(str1)!=strlen(str2))
                return 0;
   //find Frequency of str1
        for(i=0;str1[i]!='\0';i++)
        {
                if(str1[i]>='a'&&str1[i]<='z')
                {
                        num1[str1[i]-97]++;
                }
                if(str1[i]>='A'&&str1[i]<='Z')
                {
                        num1[str1[i]-65]++;
                }
        }
        //find Frequency of str2
   for(i=0;str2[i]!='\0';i++)
```

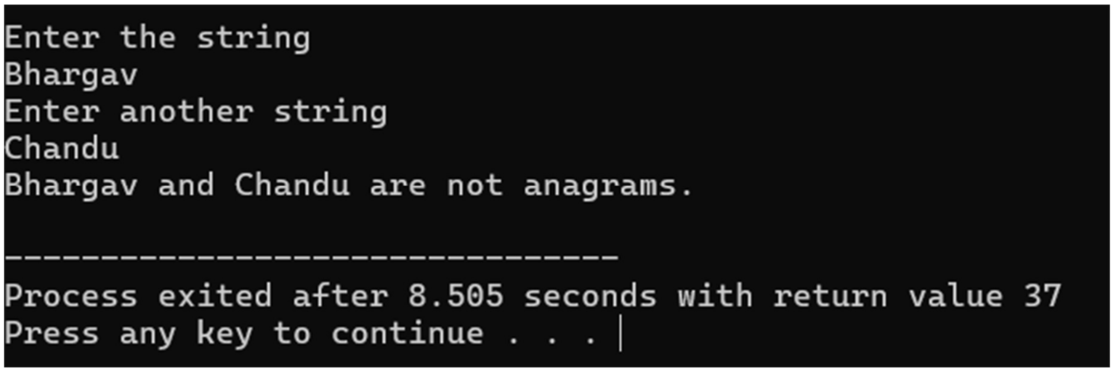
```
        {
                if(str2[i]>='a'&&str2[i]<='z')
                {
                        num2[str2[i]-97]++;
                }
                if(str2[i]>='A'&&str2[i]<='Z')
                {
                        num2[str2[i]-65]++;
                }
        }
        // check's Frequency is equal or not
   for(i=0;i<26;i++)
   {
      if(num1[i] != num2[i])
         return 0;
   }
   return 1;
}
int main()
{
   char str1[100],str2[100];
   int flag;
   printf("Enter the string\n");
   gets(str1);
   printf("Enter another string\n");
   gets(str2);
   flag = findanagram(str1,str2);
```

```
    if (flag == 1)

        printf("%s and %s are anagrams.\n",str1,str2);

    else

        printf("%s and %s are not anagrams.\n",str1,str2);

    return 0;

}
```

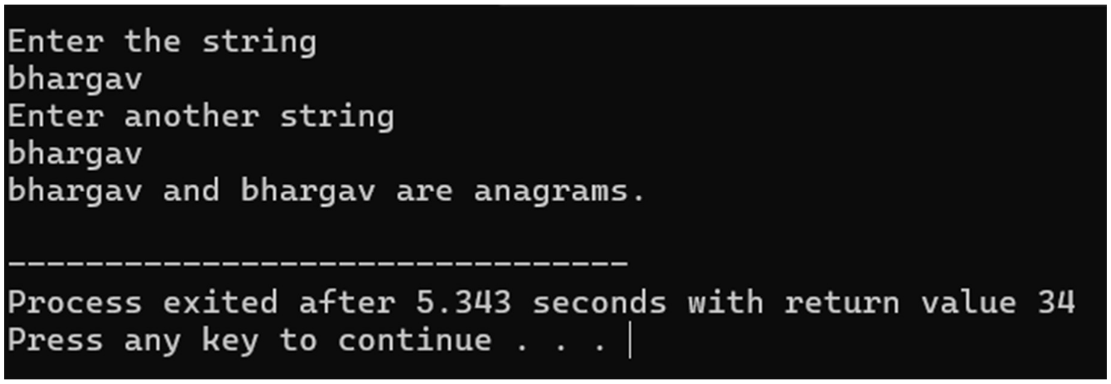
INTEGRATION AND SYSTEM TESTING

OUTPUTS

Screen Shots:



```
Enter the string
Bhargav
Enter another string
Chandu
Bhargav and Chandu are not anagrams.

--------------------------------
Process exited after 8.505 seconds with return value 37
Press any key to continue . . .
```



```
Enter the string
bhargav
Enter another string
bhargav
bhargav and bhargav are anagrams.

--------------------------------
Process exited after 5.343 seconds with return value 34
Press any key to continue . . .
```

CONCLUSION

In conclusion, the Anagram Search Algorithm developed in this project successfully identifies words that are anagrams of a given target word from a list of words. By leveraging sorting and comparison techniques, the algorithm efficiently matches anagrams while ensuring accurate results. The approach presented in this report highlights the importance of efficient data handling, algorithmic optimization, and the practical applications of anagram search in various fields such as word games, cryptography, and text processing.

The algorithm demonstrates significant performance advantages when dealing with smaller datasets and provides a solid foundation for identifying anagrams. However, as the dataset size increases, there is potential for further optimization through parallel processing or more advanced techniques like hashing.

Overall, this project not only showcases the implementation of an anagram search algorithm in C language but also serves as a stepping stone for future enhancements, such as integrating machine learning for intelligent matching or developing real-time anagram solvers for more complex scenarios. The simplicity, efficiency, and scalability of the algorithm make it a valuable tool for various real-world applications that involve word analysis and pattern recognition.

With further advancements, this system could be expanded to handle larger datasets and incorporate additional features like anagram suggestions for misspelled or incomplete words, thus enhancing its utility in dynamic environments.