Project Report no - 36

STRING EDITING

Design and Analysis of Algorithms

By

Abhiram (2010030383)

Vamsi krishna (2010030410) Sainath Reddy (2010030174)

Mitesh Chandra (2010090004)

under the supervision of

MS. DEEPTHI KALAVALA

ASSISTANT PROFESSOR



Department of Computer Science and Engineering

K L University Hyderabad,

Aziz Nagar, Moinabad Road, Hyderabad – 500 075, Telangana, India.

March, 2022

# DECLARATION

The Project Report entitled “string editing”is a record of bonafide work of Abhiram(2010030383),vamsi krishna(2010030410),Sainath(2010030174),Mitesh Chandra

(2010090004),submitted as a requirement for the completion of the course **Design and Analysis of Algorithms** in the Department of Computer Science and Engineering to the K L University, Hyderabad. The results embodied in this report have not been copied from any other Departments/University/Institute.

Signature of the Students

**CERTIFICATE**

This is to certify that the Project Report entitled “string editing” is being submitted by Abhiram(2010030383),vamsi krishna(2010030410),Sainath(2010030174),Mitesh Chandra

(2010090004) as a requirement for the completion of the course **Design and Analysis of**

**Algorithms** in the Department of Computer Science and Engineering, K L University,

Hyderabad is a record of bonafide work carried out under our guidance and supervision.

The results embodied in this report have not been copied from any other departments/ University/Institute.

**Signature of the Supervisor**

Name and Designation

**Signature of the HOD Signature of the Examiner**

# ACKNOWLEDGEMENT

# First and foremost, we thank the lord almighty for all his grace & mercy showered upon us, for completing this project successfully.

We take grateful opportunity to thank our beloved Founder and Chairman who has given constant encouragement during our course and motivated us to do this project. We are grateful to our Principal **Dr. L. Koteswara Rao** who has been constantly bearing the torch for all the curricular activities undertaken by us.

We pay our grateful acknowledgement & sincere thanks to our Head of the Department **Dr. Chiranjeevi Manike** for his exemplary guidance, monitoring and constant encouragement throughout the course of the project. We thank MS. DEEPTHI KALAVALAof our department who

has supported throughout this project holding a position of supervisor.

We whole heartedly thank all the teaching and non-teaching staff of our department without whom we won’t have made this project a reality. We would like to extend our sincere thanks especially to our parents, our family members and friends who have supported us to make this project a grand success.

# ABSTRACT:

The edit-distance between two strings is the smallest number of operations required to transform one string into the other.

The edit-distance problem for two languages is to find a pair of strings, each of which is from different language, with the minimum edit-distance.

We consider the edit-distance problem for a regular language and a context-free language and present an efficient algorithm that finds an optimal alignment of two strings, each of which is from different language. More-over, we design a faster algorithm for the edit-distance problem that only finds the minimum number of operations of the optimal alignment The string editing problem is to determine the distance between two strings as measured by the minimal cost sequence of deletions, insertions, and changes of symbols needed to transform one string into the other.

Given two strings and operations edit, delete and add, how many minimum operations would it take to convert one string to another string.

This is lavensteins algorithm which is also known as edit distance.

**TABLE OF CONTENTS**

* Introduction
* Literature Survey
* Hardware & Software requirements
* Functional & Non-functional requirements
* Proposed System
* Implementation
* Results Discussion
* Conclusion and Future Work
* References

**INTRODUCTION:**

The string editing problem is to determine the distance between two strings as measured by the minimal cost sequence of deletions, insertions, and changes of symbols needed to transform one string into the other.

A pre-requisite for applying Dynamic Programming, to solve a problem, is to demonstrate that the solution to the original problem can be found by using the solutions to the sub-problems.

The approach here is somewhat simple and initiative . Consider the strings ‘a’ and ‘b’ for their characters :

If the last characters of both strings are the same, then the edit distance is equal to the edit distance of the same two strings, up to their second-to-last character.

If the last character is different, then the edit distance is equal to the minimum of the cost of inserting, deleting, or replacing the last character of string a.

## **Literature Survey :**

|  |  |  |
| --- | --- | --- |
| NAME | TOPIC | DESCRIPTION |
| A.V. Aho, J.D. Ullman, D.S. Aho, J.D. Hirschberg | the Bounds on the complexity of the longest common subsequence problem | It tells that the problem of finding a longest common subsequence of two strings is discussed. This problem arises in data processing applications such as comparing two files and in genetic applications such as studying molecular evolution. The difficulty of computing a longest longest common subsequence of two strings is examined using the decision tree model of computation, in which vertices represent “equal - unequal” comparisons |
| HUNT. J W, AND SZYMANSKI | algorithms for finding the longest common subsequence of two sequences | An algorithm for this problem is presented which has a running time of O((r +n) log n), where r is the total number of ordered pairs of positions at which the two sequences match. Thus in the worst case the algorithm has a running time of O(n2 log n). However, for those applications where most positions of one sequence match relatively few positions in the other sequence, a running time of O(n log n) can be expected |
| Philip Klein, Srikanta Tirthapura , Daniel Sharvit | The string-to-string correction problem | tells that the *string-to-string correction problem* is to determine the distance between two strings as measured by the minimum cost sequence of “edit operations” needed to change the one string into the other. The edit operations investigated allow changing one symbol of a string into another single symbol, deleting one symbol from a string, or inserting a single symbol into a string |

## **HARDWARE & SOFTWARE REQUIREMENTS :**

hardware requirements:

* 2-core processor
* 2-gb ram
* 20-gb hard disk

software requirements:

* vscode
* windows

## **PROBLEM STATEMENT:**

Given two strings str1 and str2, and the task is to find minimum number operations required to convert string str1 into str2.

Input – Output Data for the Algorithm Str\_1 : This contains the first string.

Str\_2 : This contains the second string.

N : This contains the size of the first string.

M : This contains the size of the second string.

Solution : This is used to store the number of operations required.

The is a problem to measure how much two strings are different from one another by counting the minimum number of operations required to convert one string into the other. Edit distance problem can be solved by many different approaches . But the most efficient approach to solve the Edit distance problem is Dynamic programming approach which takes the O(N \* M) time complexity, where N and M are sizes of the strings.

Edit distance has different definitions which uses different sets of string operations. Levenshtein distance operations is the basic set of operations which is used in Edit distance Problem.

Operation allowed are:

Delete any character from the string.

Replace any character with any other

Add any character into any part of the string.

example:

Input and Output of the Example

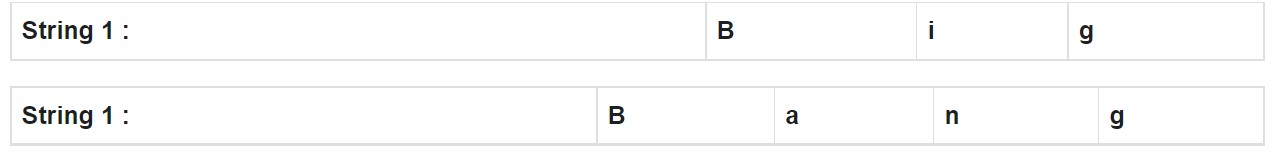
Given two strings str1 = “Big” and str2 = “Bang” of size of N = 3 and N = 3 respectively

The task is to find minimum number operations required to convert string str1 into str2.

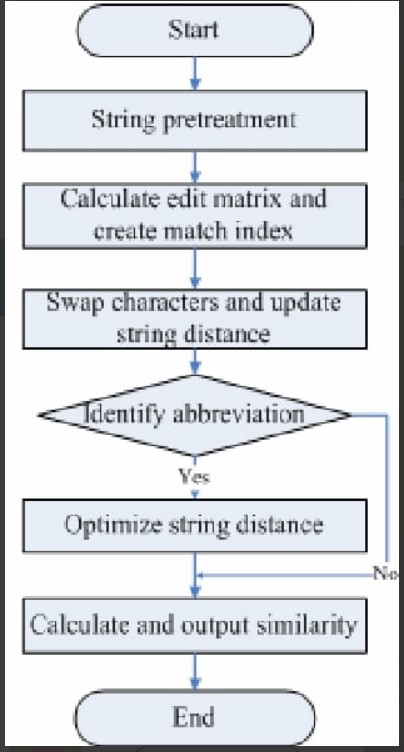
Solution of the Edit Distance Problem Example

Solution of the above example using the Dynamic programming approach.

Given data are :



**Flowchart:**



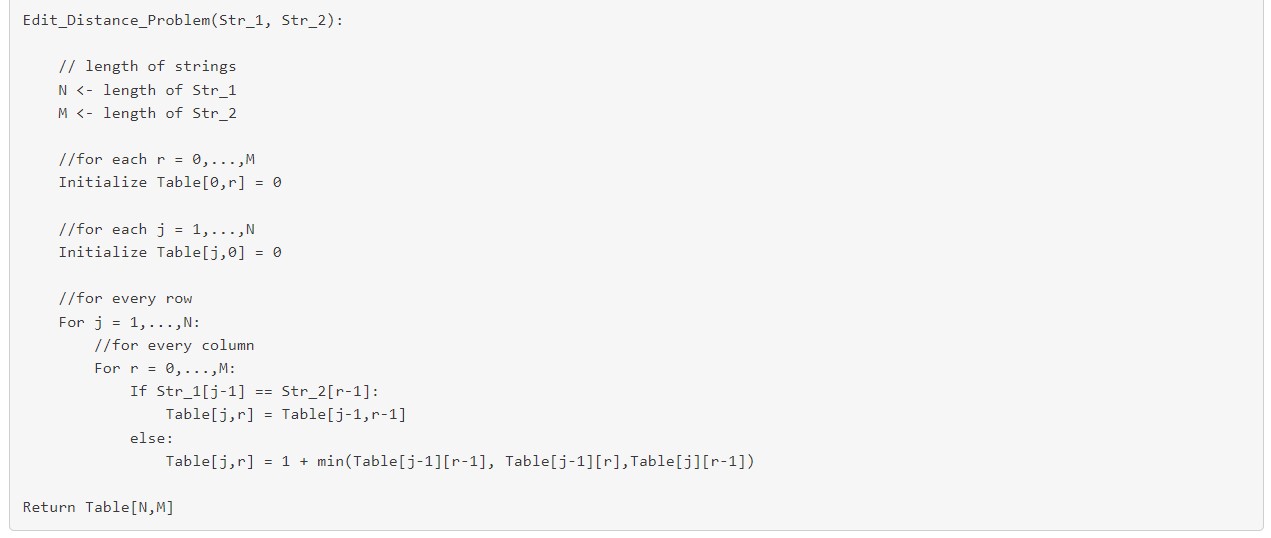
**FUNCTIONAL & NON-FUNCTI0NALREQUIREMENTS:**

This is very fast to use product as compared to others as it is relevant to string matching such as patten recognition , error correction , and molecular genetics.

In nlp , levenstein(edit distance) is used for suggesting spelling corrections , detecting plagiarism , For example, if a word is misspelt as "lighting", Levenshtein Distance can suggest that "lighting" is most similar, which helps the writer correct the spelling . It's also been used to cluster words that share a root word .

Levenshtein Distance in fact started in signal processing, in particular, to see how errors in communications systems can be corrected. Another application is speech recognition. A perfect match of an audio signal is impossible and edit distance can find the most suitable match .

**Algorithm for edit distance:**



**PROCEDURE:**

1)Create a empty table where First column represents the String 1 and First Row represents the String 2 with additional Value( empty value) in both.

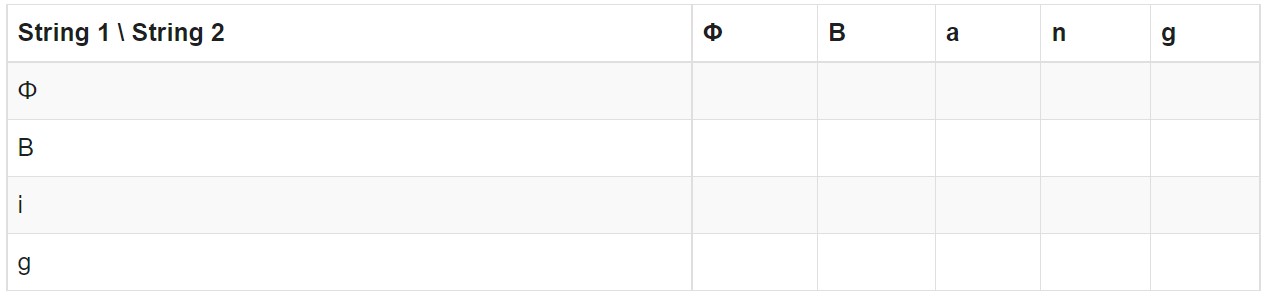
2)Let us start filling the table untill one of the string is empty. We will compare “Big” to Φ and then “Bang” to Φ.

To convert Φ to Φ, we need no operation so value is 0.

To convert B to Φ, we need 1 operation of modify, so value is 1.

To convert i to Φ, we need 2 operation of modify and insert, so value is 2. for g to Φ, value is 3.

Similarly for Bang, values will be 0, 1, 2, 3, 4.

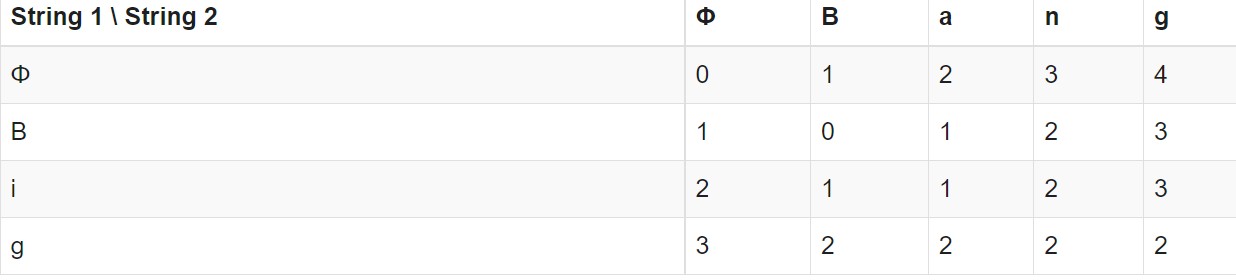


3.Now check the each character of String 1 with String 2. And update the table according to approach.

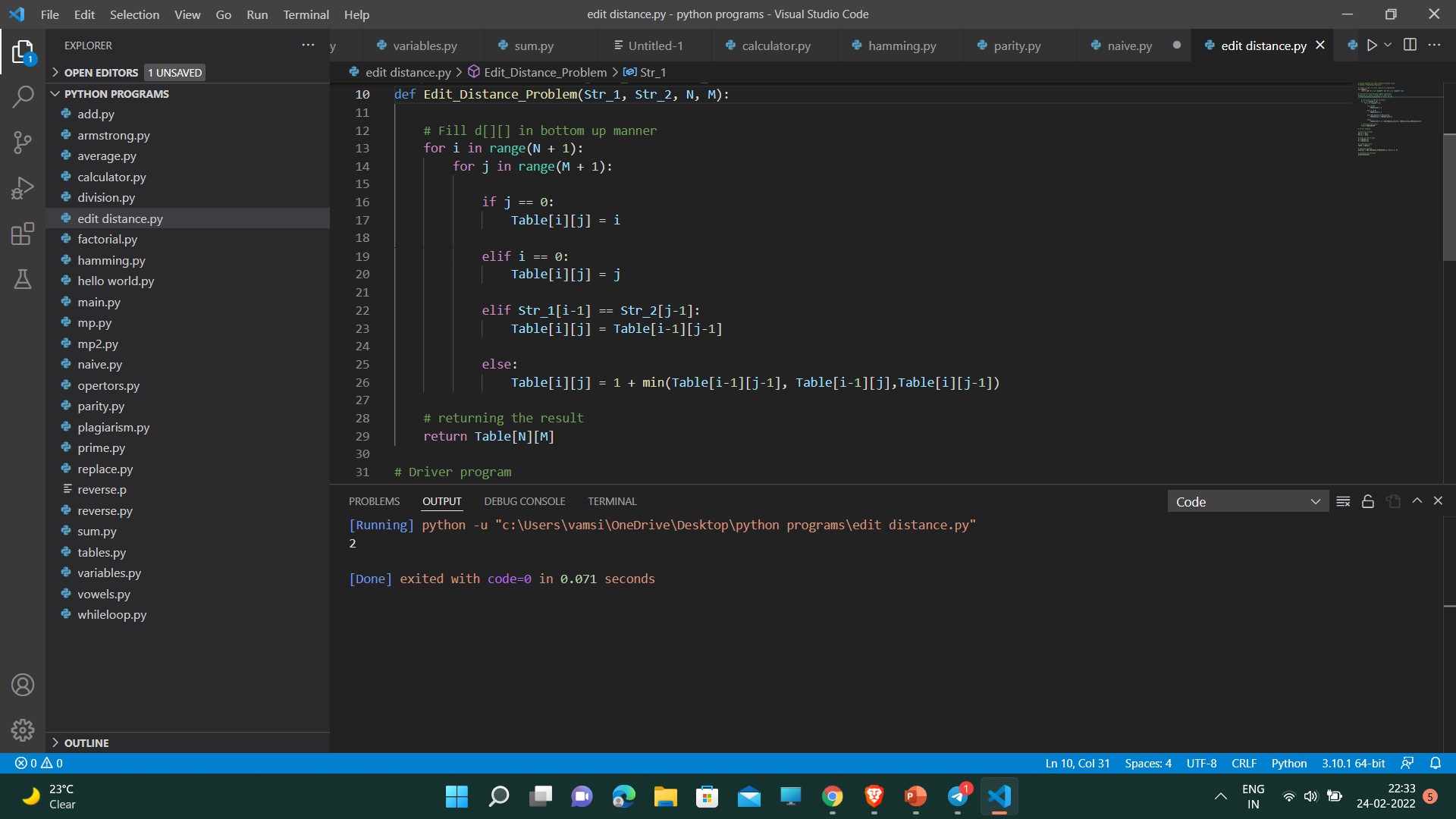
If str\_1[i-1] == str\_2[j-1]

Table [I , j] = table[i-1 , j-1] Else:

Table [I , j] = 1+min(table[i-1] , [j-1] , table [i] [j-1])



**Implementation and result:**



In the implementation part we have used the python language which is the most versatile and robust high level language. The time complexity of the above solution is exponential. In the worst case, we may end up doing O(3m) operations. The worst case happens when none of the characters of two strings match. Below is a recursive call

diagram for worst case.we can see that many subproblems are solved,again and again for example eD(2,2) is called three times since same subproblems are called again,this problem has overlapping subproblems property,so Edit distance problem has both properties of a dynamic programming problem.like other typical dynamic programming problems,recomputations of same subproblems can be avoided by constructing a temporary array that stores results of the subproblem so,this would be the entire implementation of the string editing

**conclusion and future work:**

string edit distance is a flexible error-tolerant mechanism to measure distance between two strings which has been widely applied in the correction of the words for example auto correcting feature in the mobile phones..etc this would be one of the algorithms used for the edit distance so, advantages and disadvantages of these algorithms are uncovered by comparing them experimentally and theoretically.Many ways of searching for least expensive edit sequence have been used previously. The search strategy should be consistent with the method of similarity comparison and the definition of edit cost,

instead of the best one in theory. so,we try to implement in the real life situation

**References:**

1. Umeyama S (1988) An eigendecomposition approach to weighted graph matching problems. IEEE Trans Pattern Anal Mach

Intell 10(5):695–703

1. Bunke H (2000) Recent developments in graph matching. In: Proceedings of IEEE international conference on pattern recognition, Barcelona, pp 117–124
2. Caelli T, Kosinov S (2004) An eigenspace projection clusteringmethod for inexact graph matching. IEEE Trans Pattern Anal Mach Intell 26(4):515–519
3. Cross ADJ, Wilson RC, Hancock ER (1997) Inexact graphmatching using genetic search. Pattern Recognit 30(7):953–970 5. Wagner RA, Fischer MJ (1974) The string-to-string correction problem. J ACM 21(1):168–173
4. Pavlidis JRT (1994) A shape analysis model with applications toa character recognition system. IEEE Trans Pattern Anal Mach

Intell 16(4):393–404

1. Wang Y-K, Fan K-C, Horng J-T (1997) Genetic-based search forerror-correcting graph isomorphism. IEEE Trans Syst Man

Cybern B Cybern 27(4):588–597

1. Sebastian TB, Klien P, Kimia BB (2004) Recognition of shapesby editing their shock graphs. IEEE Trans Pattern Anal Mach

Intell 26(5):550–571

1. He L, Han CY, Wee WG (2006) Object recognition and recoveryby skeleton graph matching. In: Proceedings of IEEE international conference on multimedia and expo, Toronto, pp 993–996

*THANK YOU*