

A Project Report  
on  
**FORGOTTEN PASSWORD ANALYSIS**  
Design and Analysis of Algorithms

By

GANDE SAITEJA (2010030055)

GILLA SAMANTH (2010030272)

K. TRIBHUVAN (2010030402)

MD ADNAN (2010030236)

Under the supervision of

**Ms. P. Sree Lakshmi**

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 “FORGOTTEN PASSWORD ANALYSIS” is a record of bonafide work of Gande Sai teja (2010030055), Gilla Samanth(2010030272), Md.Adnan (2010030236) and K. Tribhuvan (2010030402) submitted in partial fulfillment for the award of B. Tech 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.

GANDE SAITEJA (2010030055)

GILLA SAMANTH (2010030272)

K. TRIBHUVAN (2010030402)

MD. ADNAN (2010030236)

## **CERTIFICATE**

This is to certify that the Project Report entitled “FORGOTTEN PASSWORD ANALYSIS” is being submitted by GANDE SAITEJA bearing Regd. No. 2010030055, GILLASAMANTH bearing Regd. No. 20100300272, MD. ADNAN bearing Regd. No. 2010030236 and Mr. K. TRIBHUVAN bearing Regd. No. 2010030402 submitted in partial fulfillment for the award of B.Tech in Computer Science and Engineering to the 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 department/ University/ Institute.

**Signature of the Supervisor**

MS. P. Sree Lakshmi

Assistant Professor

**Signature of the HOD**

**Signature of the External 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 a 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 acknowledgment & sincere thanks to our Head of the Department **Dr. Chiranjeevi Manike** for her exemplary guidance, monitoring, and constant encouragement throughout the course of the project. We thank **Ms. P. Sree Lakshmi** of our department has supported us throughout this project by holding the position of supervisor.

We wholeheartedly 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 our friends who have supported us to make this project a grand success.

## ABSTRACT

Given two strings. We have to find the minimum number of steps to convert A string to B string. We have three operations to perform it.

- 1) Insert a character
- 2) Delete character
- 3) Replace a character

By using this three operations we have to convert string A to string B

The algorithm we used in these string editing concept we used Levenshtein algorithm to calculates the least number of edit operations that are necessary to modify one string to obtain another string. The most common way of calculating this is by the dynamic programming approach. 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. The longest common subsequence problem can be viewed as a special case. Wagner and Fischer proposed an algorithm that runs in time  $O(nm)$ , where  $n, m$  are the lengths of the two strings. In the present paper, it is shown that if the operations on symbols of the strings are restricted to tests of equality, then  $O(nm)$  operations are necessary (and sufficient) to compute the distance.

## TABLE OF CONTENTS

S.NO	TITLE	Page Number
1	INTRODUCTION	8
	1.1 About Dynamic programing	8
	1.2 About String Editing	8
2	LITERATURE SURVEY	9-11
	2.1 Existing System	9
	2.2 Survey Table	10
3	HARDWARE & SOFTWARE REQUIREMENTS	11
	3.1 Hardware Requirements	11
	3.2 Software Requirements	11
4	FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS	11
	4.1 Functional Requirements	12
	4.2 Non-Functional Requirements	12
5	PROPOSED SYSTEM	13-15
	5.1 Model	13
	5.2 Algorithms & Techniques	14
	5.3 Complexity Analysis	15
6	IMPLEMENTATION	16-18
	6.1 Workflow	16
	6.2 Explanation	17
	6.3 Code	18
7	RESULTS DISCUSSION	19-20
8	CONCLUSION AND FUTURE WORK	21
	8.1 Conclusion	21
	8.2 Future Work	21
9	REFERENCES	22

## List of Figures

Chapter No.	Figure No.	Figure Name	Page Number
2	2.1.1	Recursion	9
2	2.1.2	Dynamic	9
6	6.1.1	Flow chart	16
7	7.1	Data Input	19
7	7.2	Data Stored	19
7	7.3	Login Page	19
7	7.4	Final page	20

## List of Tables

Chapter No.	Table No.	Table Name	Page Number
2	2.2.3	Survey Table	10

# 1. INTRODUCTION

As now a days in today's life people are not able to remember the passwords of there applications due to more number of applications and sites have good security .so to get correct passwords through the guess of password enter to the model designed to know the minimum possible changes to get right password. This is possible through dynamic string editing .

## 1.1) What is Dynamic programming?

Dynamic Programming it is mainly an optimization over plain recursion. dynamic programming is both a mathematical optimization method and a computer programming method. The method was developed by Richard Bellman in the 1950s and has found applications in numerous fields, from aerospace engineering to economics. In both contexts it refers to simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner. While some decision problems cannot be taken apart this way, decisions that span several points in time do often break apart recursively. Likewise, in computer science, if a problem can be solved optimally by breaking it into sub-problems and then recursively finding the optimal solutions to the sub-problems, then it is said to have optimal substructure.

If sub-problems can be nested recursively inside larger problems, so that dynamic programming methods are applicable, then there is a relation between the value of the larger problem and the values of the sub-problems.

## 1.2) What is Edit Distance?

An optimal transcript is an edit transcript with the minimal number of edit operations for transforming one string into another. The edit distance problem entails computing the edit distance between two strings along with an optimal transcript.

Note: optimal transcripts may not be unique.



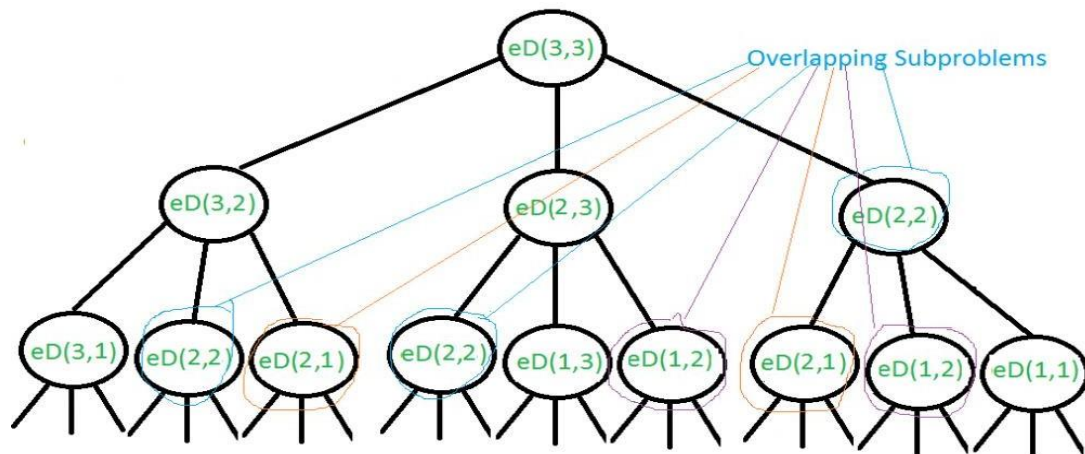
## 2. LITERATURE SURVEY

### 2.1.1) Recursion

All the characters of both the strings are traversed one by one either from the left or the right end and apply the given operations.

Time Complexity:  $O(3^{(N * M)})$ , where  $N$  and  $M$  is the length of the first and second

Space Complexity:  $O(N + M)$ , where  $N$  and  $M$  is the length of the first and second string.



Worst case recursion tree when  $m = 3$ ,  $n = 3$ .  
Worst case example  $str1 = "abc"$   $str2 = "xyz"$

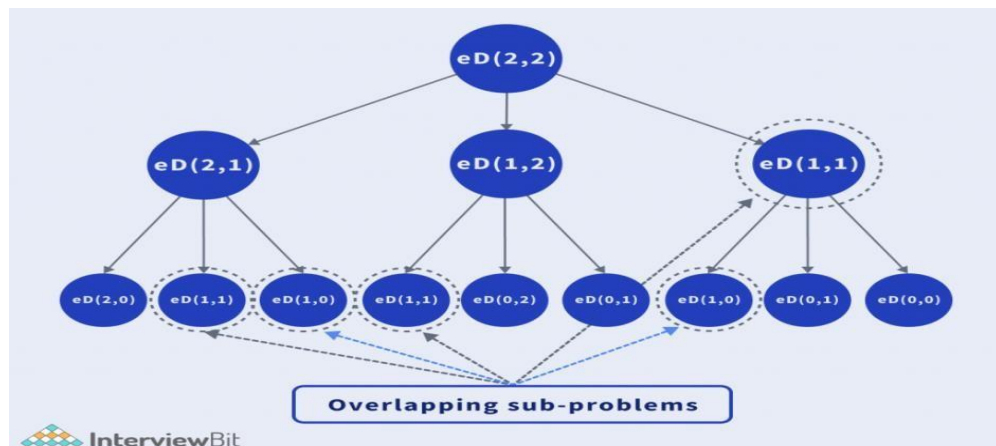
Fig: 2.1.1 RECURSION

### 2.1.2) Dynamic programming

The idea is to use a dynamic programming approach here. The tabulation method is the most efficient method to solve this problem. As stated earlier, since the problem has overlapping subproblems, many of the calculations are repeated.

Time Complexity:  $O(N * M)$ , where  $N$  and  $M$  is the length of the first and second string.

Space Complexity:  $O(N * M)$ , where  $N$  and  $M$  is the length of the first and second string.



## 2.2) LITERATURE SURVEY TABLE

S . NO	AUTHORS	TITLE	PUBLISHING
1	Mohamed El Bachir Menai, Nailah Salah Al	Similarity Detection in Java Programming Assignments	August 24–27, 2018
2	Mohamed El Bachir Menai, Manar BagaiS	APlag: A Plagiarism Checker for Arabic Texts	-August 3-5, 2018
3	Shauna D. Stephens	USING METRICS TO DETECT PLAGIARISM	June 2019
4	Lannan Luo Jiang Ming Dinghao Wu Peng Liu Sencun Zhu	Similarity Comparison with Applications to Software Plagiarism Detection	July 2017
5	Asim M. El Tahir Ali, Hussam M. Dahwa	Overview and Comparison of Plagiarism Detection Tools	August 2018

**Fig: 2.2.3( Survey Table)**

### **3. HARDWARE & SOFTWARE REQUIREMENTS**

#### **3.1 HARDWARE REQUIREMENTS:**

Operating System: Any operating system

Supporting System: 64-Bit Operating System, x64-based

processorProcessor: Intel® Core i5 7<sup>th</sup> Gen 2.50GHz

RAM: 8 GB

#### **3.2 SOFTWARE REQUIREMENTS:**

Software: PyCharm Community

Edition Programming Language:

Python 3.7

## **4. FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS**

### **4.1 Functional Requirements**

Make sure you are working in the software which supports the desired language such as Python, Java, C, etc. where in this project we have used **PyCharm Community Edition**

### **4.2 Non-Functional Requirements**

- Reliability
- Performance
- Maintainability
- Serviceability
- Data Integrity
- Usability
- Recoverability

## 5. PROPOSED SYSTEM

Dynamic Programming is also used in optimization problems. Like divide-and-conquer method, Dynamic Programming solves problems by combining the solutions of subproblems. Moreover, Dynamic Programming algorithm solves each sub-problem just once and then saves its answer in a table, thereby avoiding the work of re-computing the answer every time.

Two main properties of a problem suggest that the given problem can be solved using Dynamic Programming. These properties are overlapping sub-problems and optimal substructure. **Overlapping Sub-Problems** Similar to Divide-and-Conquer approach, Dynamic Programming also combines solutions to sub-problems. It is mainly used where the solution of one sub-problem is needed repeatedly. The computed solutions are stored in a table, so that these don't have to be re-computed. Hence, this technique is needed where overlapping sub-problem exists. For example, Binary Search does not have overlapping sub-problem. Whereas recursive program of Fibonacci numbers have many overlapping sub-problems. **Optimal Sub-Structure** A given problem has Optimal Substructure Property, if the optimal solution of the given problem can be obtained using optimal solutions of its sub-problems. For example, the Shortest Path problem has the following optimal substructure property –

The standard All Pair Shortest Path algorithms like Floyd-Warshall and Bellman-Ford are typical examples of Dynamic Programming.

## 5.1) ALGORITHMS & TECHNIQUES

### LEVENSHTEIN ALGORITHM

- The Levenshtein algorithm calculates the least number of edit operations that are necessary to modify one string to obtain another string. The most common way of calculating this is by the dynamic programming approach:
- A matrix is initialized measuring in the (m, n) cell the Levenshtein distance between the m-character prefix of one with the n-prefix of the other word.
- The matrix can be filled from the upper left to the lower right corner.
- Each jump horizontally or vertically corresponds to an insert or a delete, respectively.
- The cost is normally set to 1 for each of the operations.
- The diagonal jump can cost either one, if the two characters in the row and column do not match else 0, if they match. Each cell always minimizes the cost locally.
- This way the number in the lower right corner is the Levenshtein distance between both words.

### STEPS: -

Step-1: Initialize a 2D dp, where  $dp[i][j]$  denotes the edit distance of the length(i+1)th of A and (j + 1)th length of B.

Step-2: The recurrence relation is as follows:

Step-3: If current character of both the strings are same:  $dp[i][j] = dp[i - 1][j - 1]$

Step-4: If current character of both the strings are different:  $dp[i][j] = 1 + \min(dp[i - 1][j - 1], dp[i - 1][j], dp[i][j - 1])$

## **5.2) Complexity Analysis**

### **5.2.1) Time Complexity**

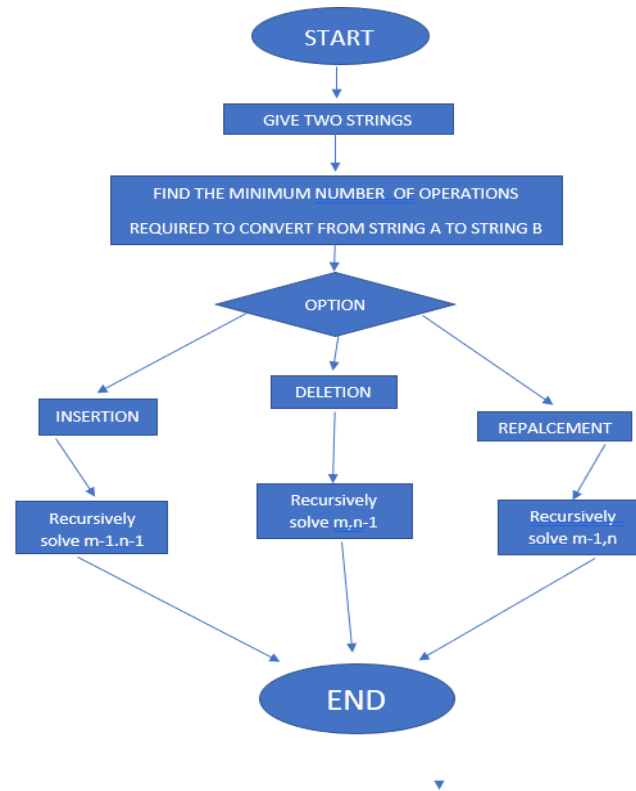
The time complexity of above solution is exponential. In worst case, we may end up doing  $O(3^m)$  operations. The worst case happens when none of characters of two strings match. Below is a recursive call diagram for worst case. We can see that many subproblems are solved, again and again, for example,  $eD(2, 2)$  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(DP) problems, recomputations of same subproblems can be avoided by constructing a temporary array that stores results of subproblems.

### **5.2.2) SPACE COMPLEXITY**

In the above-given method we require  $O(m \times n)$  space. This will not be suitable if the length of strings is greater than 2000 as it can only create 2D array of  $2000 \times 2000$ . To fill a row in DP array we require only one row the upper row. For example, if we are filling the  $i = 10$  rows in DP array we require only values of 9th row. So we simply create a DP array of  $2 \times \text{str1 length}$ . This approach reduces the space complexity.

## 6. IMPLEMENTATION

### 6.1 Workflow



**Fig: 6.1.1 Flow Chart**



## 6.2 Explanation

- The code starts by declaring the strings that will be used in the function.
- The variable n is set to an input from the user, and password is a dictionary of all possible programs sorted by their length.
- The next step is to create a list called prgm which contains all of the program numbers and lengths for each program.
- Then user details will be stored in the database for future process.
- Django database where all the details of the users will be stored such as userName, Email and Password so that user can login.
- Here user trying to login but he forgot his password so here is the situation where our project forgotten password analysis works so when user enters his password it will check the password which user enters with the password which was stored in the website
- So here in these situation the user have entered mismatch of 4 letters compared to his original password so there is a 90% chances for the user to recap 4 letters in his password.
- So here in these situation the user have entered mismatch of 1 letters compared to his original password. So the user decreased from mismatch of 4 letters to mismatch of 1 letter. So now chances for the user to recap 1 letters in his password is easier.
- So finally user will be login into his account successfully.....

### 6.3) Code

```
def register(request):
    if request.method == 'POST':
        uname = request.POST['uname']
        email = request.POST['email']
        password = request.POST['password']
        cpassword = request.POST['cpassword']

        if password==cpassword:
            user = Students(username=uname,password=password)
            user.save()
            print("registered")
            messages.info(request,'Registration successfull..!')
            return redirect('register')
        else:
            print('passwords do not match :(')
            messages.info(request,'Passwords does not match')
            return redirect('register')
        else:
            return render(request , 'register.html')

def login(request):
    if request.method == 'POST':
        uname = request.POST['uname']
        password = request.POST['password']

        u = Students.objects.get(username__exact=uname)
        # if u:
        #     auth.login(request,user)
        #     passw = u.password

        print(u.password)
        if password == u.password:
            print('login success')
            return
        if difference>4 :
            messages.info(request,'Entered password is completely invalid')
            return redirect('login')
        if x >= 1 and x <= 3:
            messages.info(request,f'your almost there there is amismatch of only')
            return redirect('login')
        elif x >= 4 and x <= 6:
            messages.info(request,f"COME ON DONT GIVE UP THERE IS ONLY MISMATCH OF password")
            return redirect('login')
```

## 7. RESULTS DISCUSSION

This is the first page which is called as a Registration page where user will be registered using his Name, Email and password

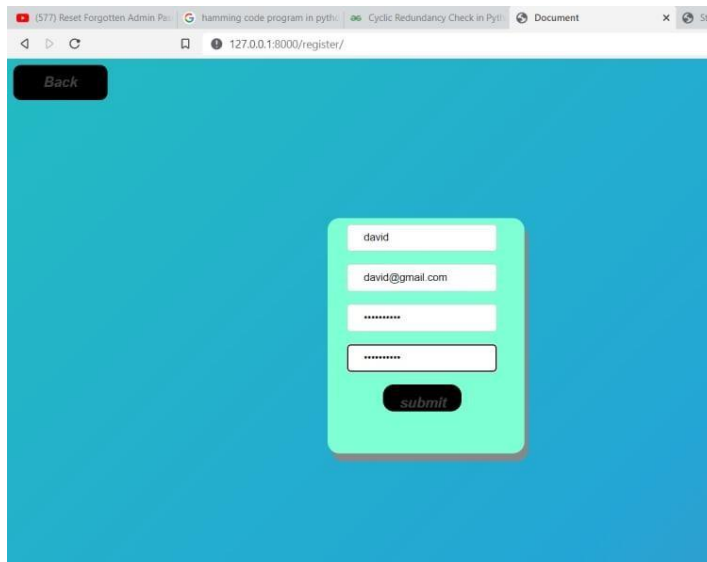


Fig: 7.1 Data Input

This the Django database where all the details of the users will be stored such as userName, Email and Password so that user can login in

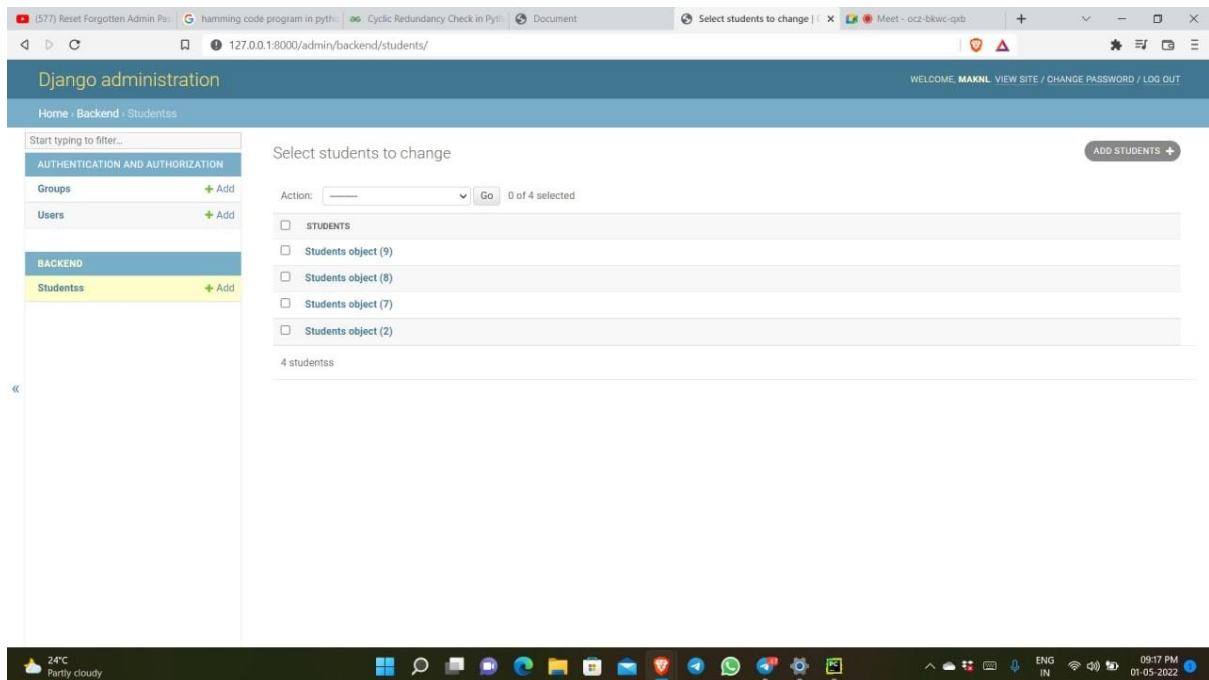


Fig: 7.2 Data Stored

Here user trying to login but he forgot his password so here is the situation where our project forgotten password analysis works so when user enters his password it will check the password which user enters with the password which was stored in the website

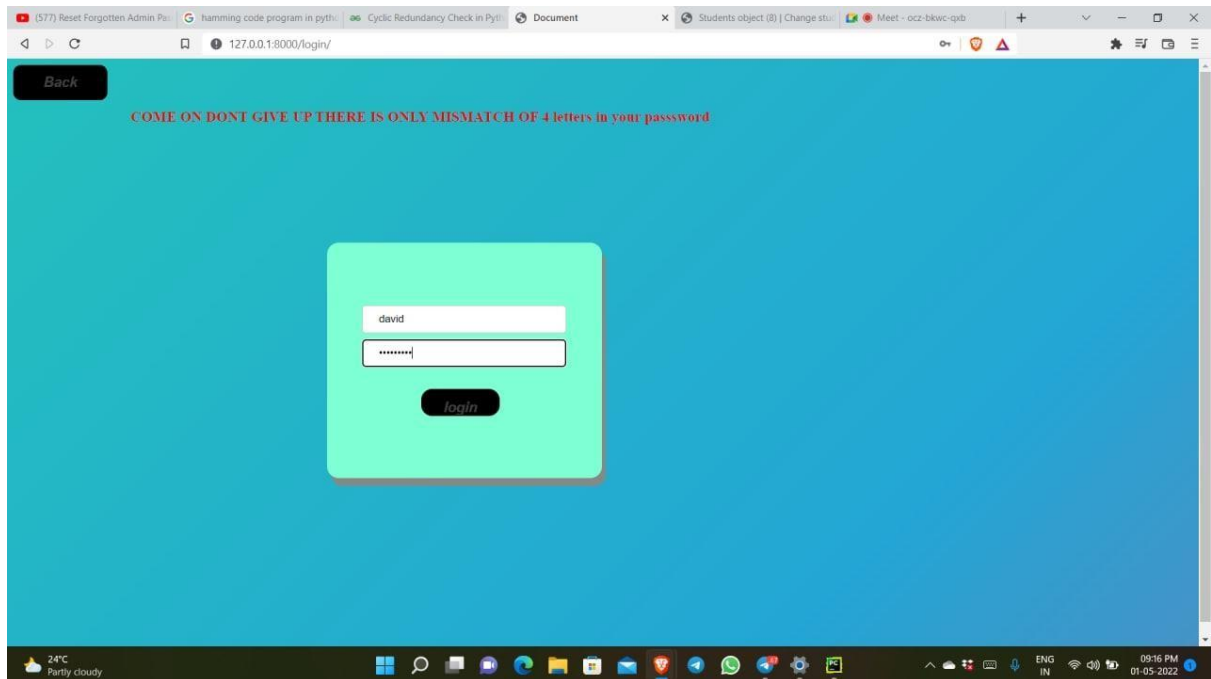


Fig: 7.3 Login Page

So finally user as login into his account successfully.....

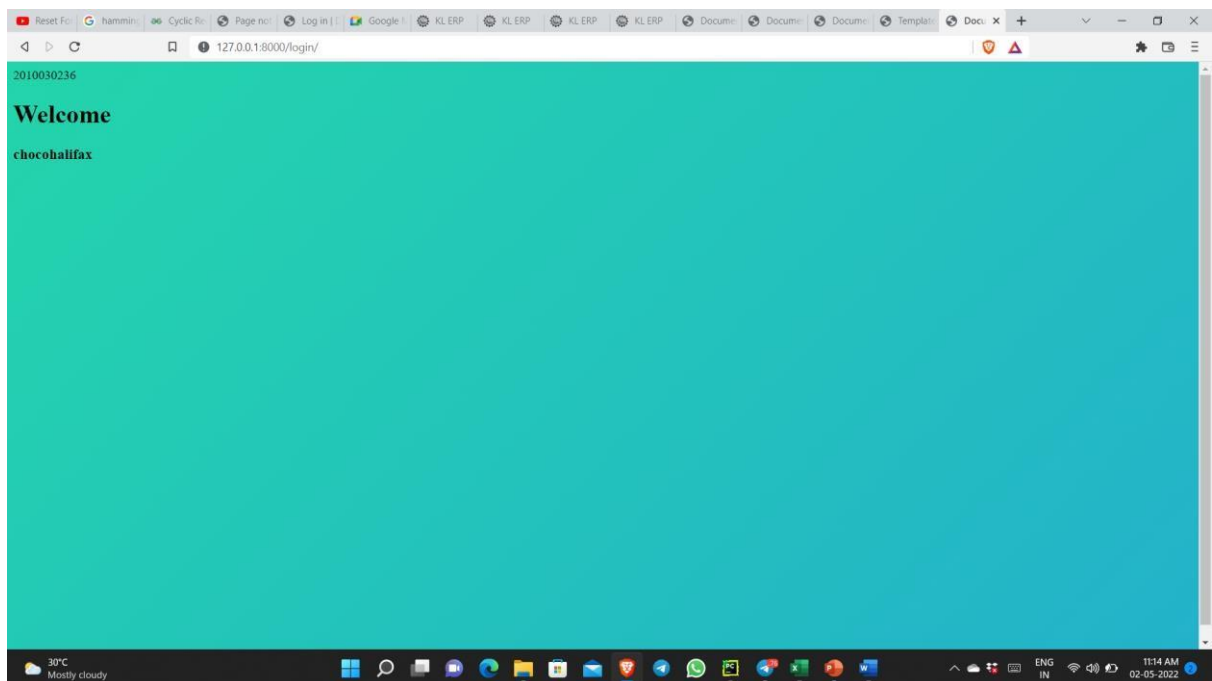


Fig: 7.3 Final Page

## 8. CONCLUSION & FUTURE WORK

### 8.1 Conclusion

In this paper, we present an efficient algorithm to compute the edit distance between a string. By the model designed webpage for forgotten password analysis using string dynamic editing-Any path from  $(n,m)$  to  $(0,0)$  corresponds to an optimal edit sequence and an optimal alignment. We recover all optimal edit sequences and alignments simply by extracting all paths from  $(n,m)$  to  $(0,0)$ . The correspondence between paths and edit sequences is one-to-one. The correspondence between paths and alignments is one-to-one. And insertion, deletion, replacing we are able to find how close entered password is to original password and it gives the number of operations need to be done.

### 8.2 Future Work

Dynamic programming can be applied to any problem that observes the principle of optimality. Roughly stated, this means that partial solutions can be optimally extended with regard to the state after the partial solution instead of the partial solution itself. For example, to decide whether to extend an approximate string matching by a substitution, insertion, or deletion, we do not need to know exactly which sequence of operations was performed to date. In fact, there may be several different edit sequences that achieve a cost of  $C$  on the first  $p$  characters of pattern  $P$  and  $t$  characters of string  $T$ . Future decisions will be made based on the consequences of previous decisions, not the actual decisions themselves.

## 9. REFERENCES

- B. S. Baker. On finding duplication and near-duplication in large software systems. In WCRE'95, pages 86–95, 1995.
- G. Balakrishnan and T. Reps. WYSINWYX: What You See Is Not What You eXecute. ACM Transactions on Programming Languages and Systems (TOPLAS), 32(6):23:1–23:84, Aug. 2010.
- G. Balakrishnan, T. Reps, D. Melski, and T. Teitelbaum. WYSINWYX: What You See Is Not What You eXecute. In Verified Software: Theories, Tools, Experiments (VSTTE), pages 202–213,
  - 2005.
- C. Cadar, D. Dunbar, and D. Engler. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In OSDI'08, 2008.
- C. Cadar and D. Engler. Execution generated test cases: How to make systems code crash itself. In SPIN'05, 2005.