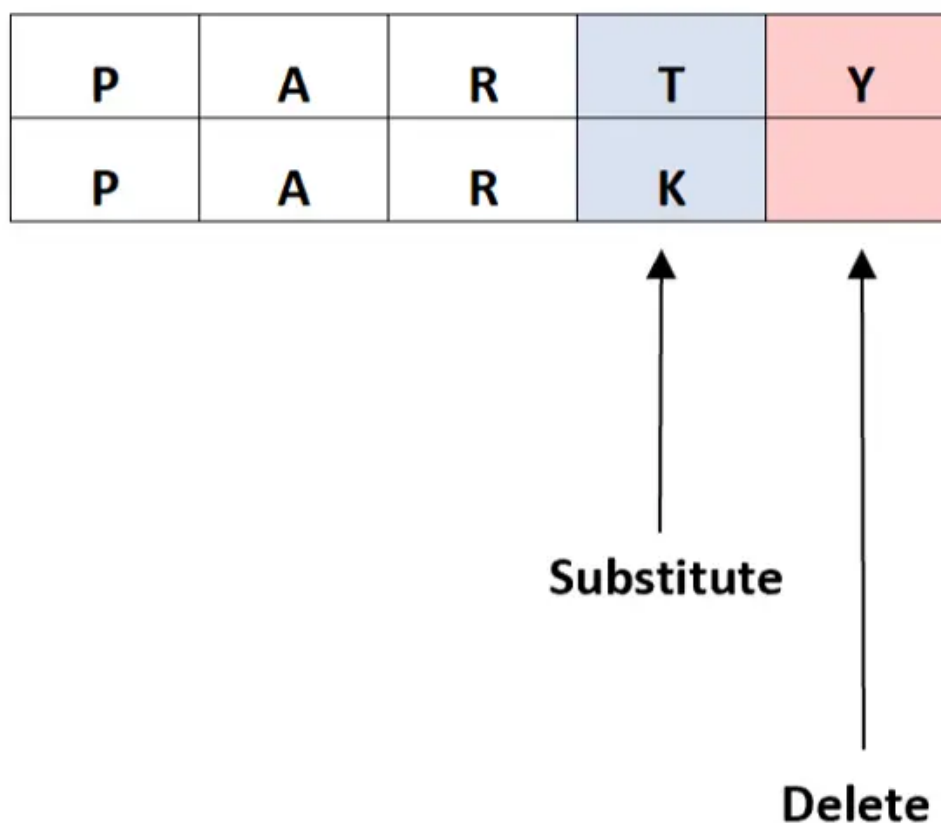




The Levenshtein distance algorithm

Free System Design Interview Course

Many candidates are rejected or down-leveled due to poor performance in their System Design Interview. Stand out in System Design Interviews and get hired in 2023 with this popular free course.

[Get Free Course](#)

The **Levenshtein distance (a.k.a edit distance)** is a measure of similarity between two strings. It is defined as the minimum number of changes required to convert string **a** into string **b** (this is done by inserting, deleting or replacing a character in string **a**). The smaller the Levenshtein distance, the

more similar the strings are. This is a very common problem in the application of [Dynamic Programming](#).



Explanation

Using sub-problems

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 intuitive. Consider the strings **a** and **b** up to their last character:

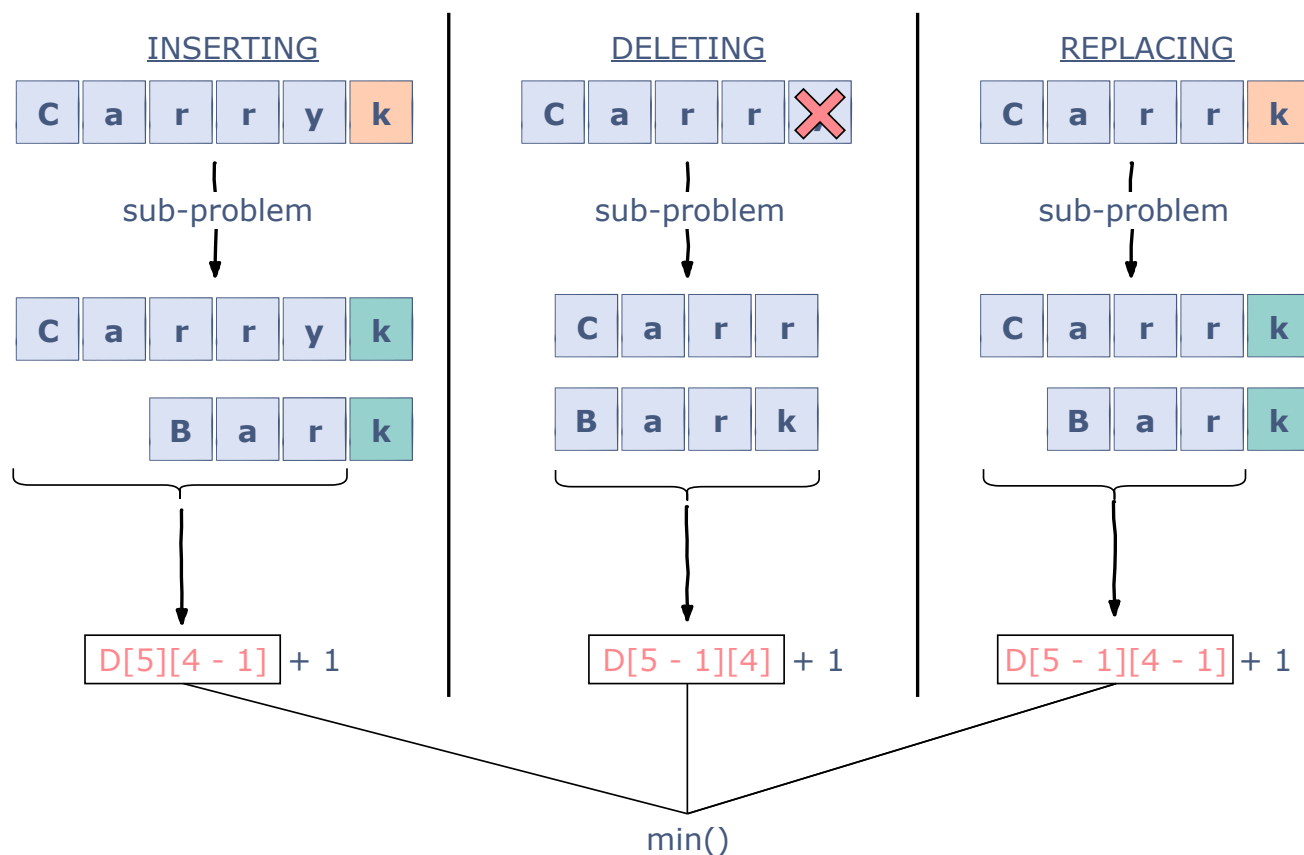
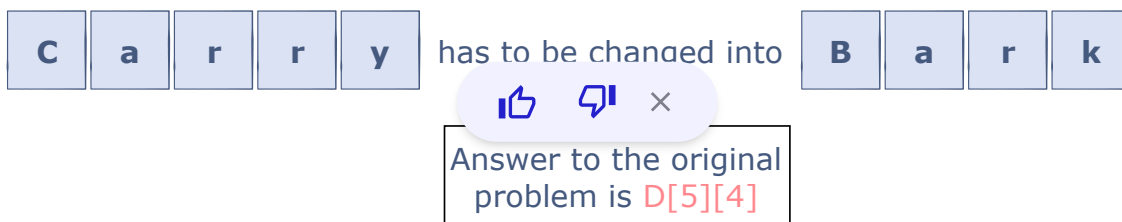
1. If the last characters of both strings are the same, then the edit distance is equal to the edit distance of the same two strings, up to their second-to-last character.
2. If the last character is different, then the edit distance is equal to the *minimum* of the cost of inserting, deleting, or replacing the last character of string **a**.

Understanding the array

Since DP utilises memory for storing the solutions to sub-problems, we will use a 2D array (**D**) for this purpose.

The value in **D[i][j]** represents the edit distance *if* we consider the strings **a** and **b**, till their **i**th and **j**th character, respectively. Therefore, the answer to the original problem will be found in **D[length of 'a'][length of 'b']**.





How sub-problems are used in Levenshtein's algorithm

Code

```

7
8 # Initialising first row:
9 for i in range(len(a) + 1):
10     D[i][0] = i
11
12 # Initialising first column:
13 for j in range(len(b) + 1):
14     D[0][j] = j
15
16 for i in range(1, len(a) + 1):
17     for j in range(1, len(b) + 1):
18         if a[i - 1] == b[j - 1]:
19             D[i][j] = D[i - 1][j - 1]
20         else:
21             # Adding 1 to account for the cost of operation
22             insertion = 1 + D[i][j - 1]
23             deletion = 1 + D[i - 1][j]
24             replacement = 1 + D[i - 1][j - 1]
25

```

RELATED TAGS

```
26 # Choosing the best option:  
27 D[i][j] = min(insertion, deletion, replacement)  
28  
29 print("Levenshtein Distance: ", D[len(a)  
30  
31  
32 License: Creative Commons-Attribution-ShareAlike  
33 4.0 (CC-BY-SA 4.0)  
34  
35  
36  
37
```

[Code in C++, Java and Python](#)

Related Courses



 Coderust

Coderust: Hacking the Coding Interview

 Beginner

[Preview](#) →




Keep Exploring

What is the difference between == and is in Python?

Algorithms for third-generation sequencing



What is
function.



ect1d()



Learn in-demand tech skills in half the time

PRODUCTS

- Learning
- CloudLabs **New**
- Onboarding
- Skill Assessments
- Projects

LEGAL

- Privacy Policy
- Cookie Policy
- Terms of Service
- Business Terms of Service
- Data Processing Agreement

RESOURCES

- Blog
- EM Hub
- Sessions
- Answers

PRICING

- For Individuals
- Free Trial

CONTRIBUTE

- Become an Author
- Become an Affiliate
- Become a Contributor

ABOUT US

- Our Team
- Careers **Hiring**
- Frequently Asked Questions
- Contact Us



Press



MORE

[GitHub Students Scholarship](#)[Course Catalog](#)[Early Access Courses](#)[Earn Referral Credits](#)[CodingInterview.com](#)

Copyright ©2023 Educative, Inc. All rights reserved.

