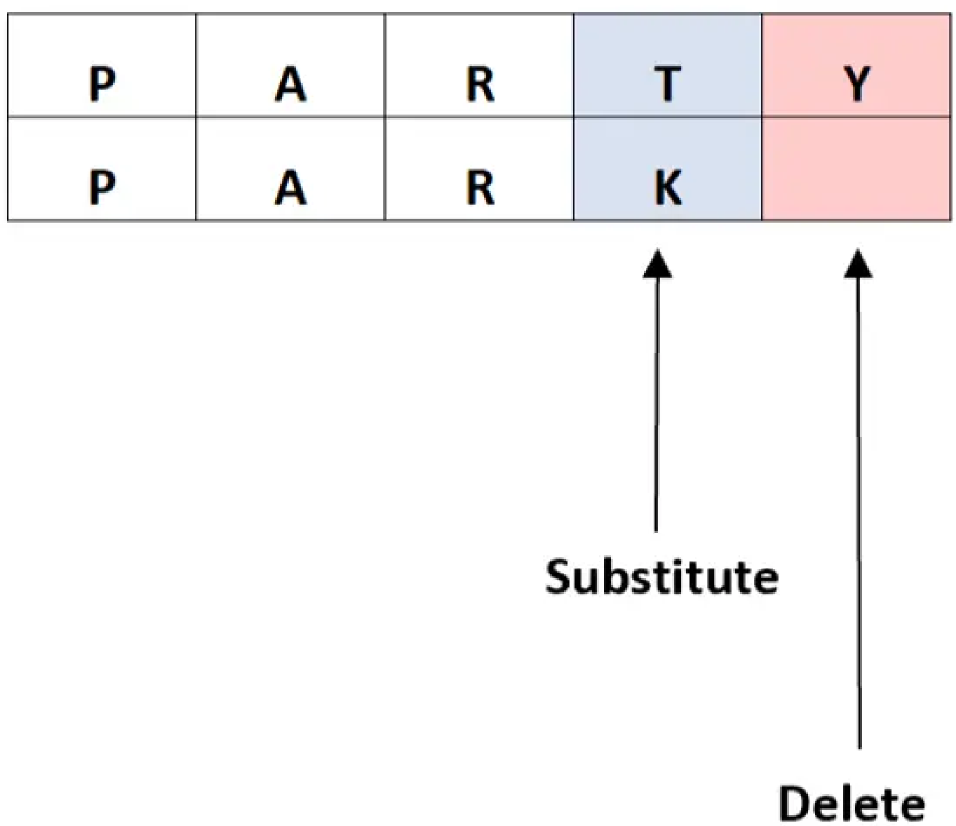
The Levenshtein distance algorithm



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 e 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.

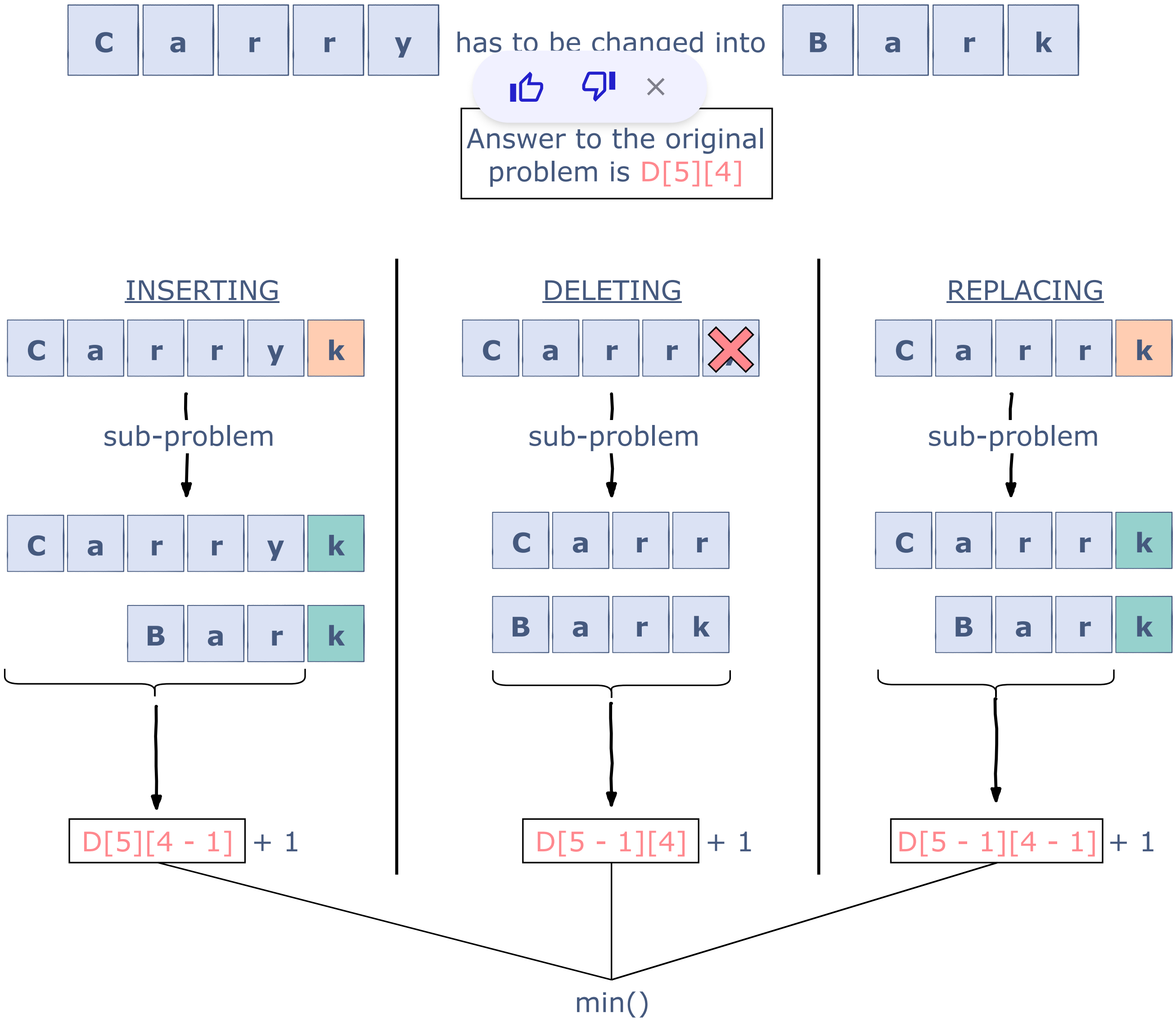
The approach here is somewhat simple and intuitive. Consider the strings `a` and `b` up to their last character:

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

```

# Initialising first row:
for i in range(len(a) + 1):
    D[i][0] = i

# Initialising first column:
for j in range(len(b) + 1):
    D[0][j] = j

for i in range(1, len(a) + 1):
    for j in range(1, len(b) + 1):
        if a[i - 1] == b[j - 1]:
            D[i][j] = D[i - 1][j - 1]
        else:
            # Adding 1 to account for the cost of operation
            insertion = 1 + D[i][j - 1]
            deletion = 1 + D[i - 1][j]
            replacement = 1 + D[i - 1][j - 1]
```

RELATED TAGS

```
        # Choosing the best option:
        D[i][j] = min(insertion, deletion, replacement)

print("Levenshtein Distance: ", D[len(a
```

python operator == is bioinformatics

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 ... ect1d() function

Press

MORE

GitHub Students Scholarship

Course Catalog

Early Access Courses

Earn Referral Credits

CodingInterview.com