Minimum Deletions in a String to make it a Palindrome

We'll cover the following

Problem Statement

Given a string, find the minimum number of characters that we can remove to make it a palindrome.

Example 1:

```
Input: "abdbca"
Output: 1
Explanation: By removing "c", we get a palindrome "abdba".
```

Example 2:

```
Input: = "cddpd"
Output: 2
Explanation: Deleting "cp", we get a palindrome "ddd".
```

Example 3:

```
Input: = "pqr"
Output: 2
Explanation: We have to remove any two characters to get a palindrome, e.g. if we
remove "pq", we get palindrome "r".
```

Solution

This problem can be easily converted to the Longest Palindromic Subsequence (https://www.educative.io/collection/page/5668639101419520/5633779737559040/5748119283171328/) (LPS) problem. We can use the fact that LPS is the best subsequence we can have, so any character that is not part of LPS must be removed. Please note that it is 'Longest Palindromic SubSequence' and not 'Longest Palindrome Substring'.

So, our solution for a given string ‘st’ will be:

```
Minimum_deletions_to_make_palindrome = Length(st) - LPS(st)
```

Code #

Let’s jump directly to bottom-up dynamic programming:

Java JS Python3 C++

```python
def find_minimum_deletions(st):
  # subtracting the length of Longest Palindromic Subsequence from the length of
  # the input string to get minimum number of deletions
  return len(st) - find_LPS_length(st)


def find_LPS_length(st):
  n = len(st)
  # dp[i][j] stores the length of LPS from index 'i' to index 'j'
  dp = [[0 for _ in range(n)] for _ in range(n)]

  # every sequence with one element is a palindrome of length 1
  for i in range(n):
    dp[i][i] = 1

  for startIndex in range(n - 1, -1, -1):
    for endIndex in range(startIndex + 1, n):
      # case 1: elements at the beginning and the end are the same
      if st[startIndex] == st[endIndex]:
        dp[startIndex][endIndex] = 2 + dp[startIndex + 1][endIndex - 1]
      else:  # case 2: skip one element either from the beginning or the end
        dp[startIndex][endIndex] = max(
          dp[startIndex + 1][endIndex], dp[startIndex][endIndex - 1])

  return dp[0][n - 1]


def main():
  print(find_minimum_deletions("abdbca"))
  print(find_minimum_deletions("cddpd"))
  print(find_minimum_deletions("pqr"))


main()
```

The time and space complexity of the above algorithm is $O(n^2)$, where ‘n’ is the length of the input string.

Similar problems

Here are a couple of similar problems:

1. Minimum insertions in a string to make it a palindrome

Will the above approach work if we make insertions instead of deletions?

Yes, the length of the Longest Palindromic Subsequence is the best palindromic subsequence we can have. Let's take a few examples:

Example 1:

```
Input: "abdbca"
Output: 1
```

Explanation: By inserting "c", we get a palindrome "a**c**bdbca".

Example 2:

```
Input: = "cddpd"
Output: 2
```

Explanation: Inserting "cp", we get a palindrome "cd**p**dpd**c**". We can also get a palindrome by inserting "dc": "cddpd**dc**"

Example 3:

```
Input: = "pqr"
Output: 2
```

Explanation: We have to insert any two characters to get a palindrome (e.g. if we insert "pq", we get a palindrome "pqr**qp**").

2. Find if a string is K-Palindromic

Any string will be called K-palindromic if it can be transformed into a palindrome by removing at most 'K' characters from it.

This problem can easily be converted to our base problem of finding the minimum deletions in a string to make it a palindrome. If the "minimum deletion count" is not more than 'K', the string will be K-Palindromic.