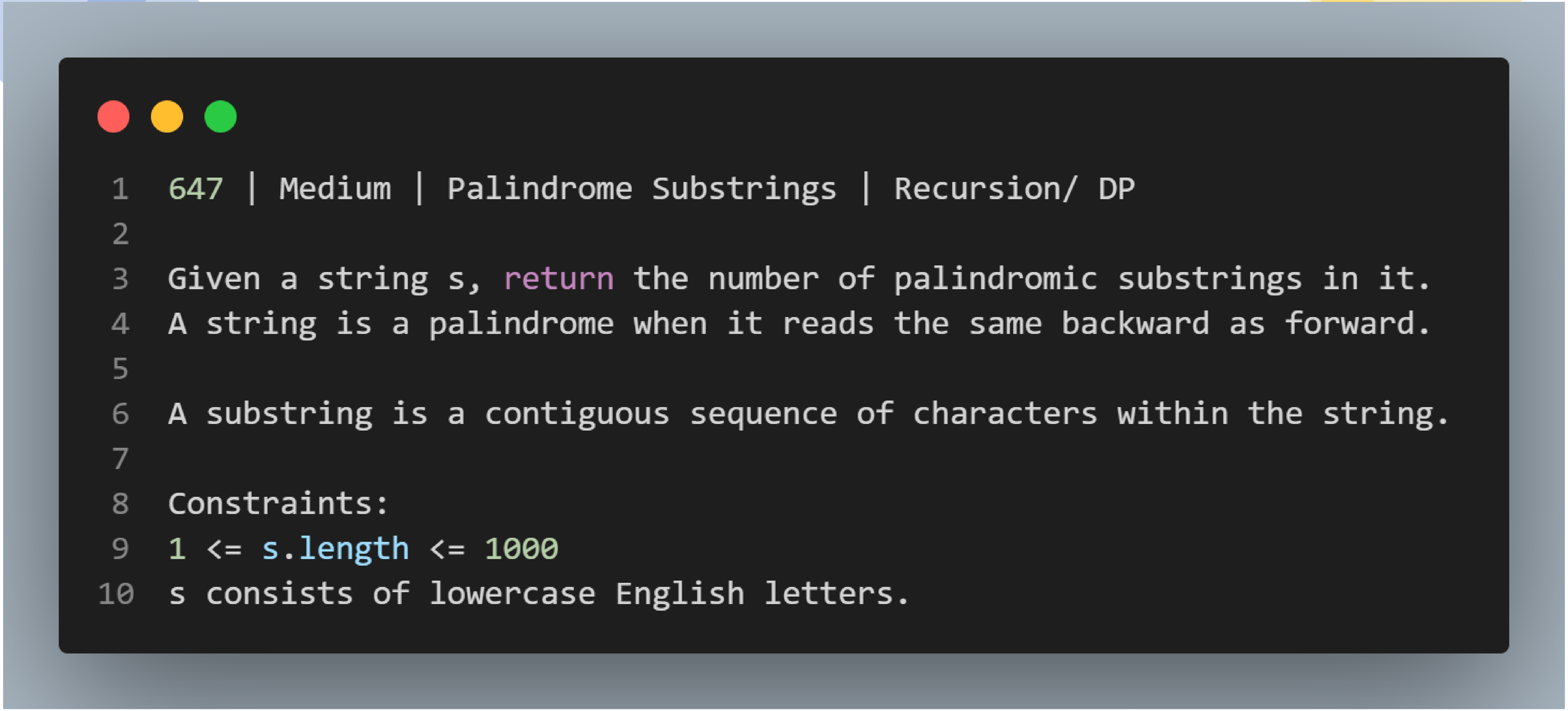
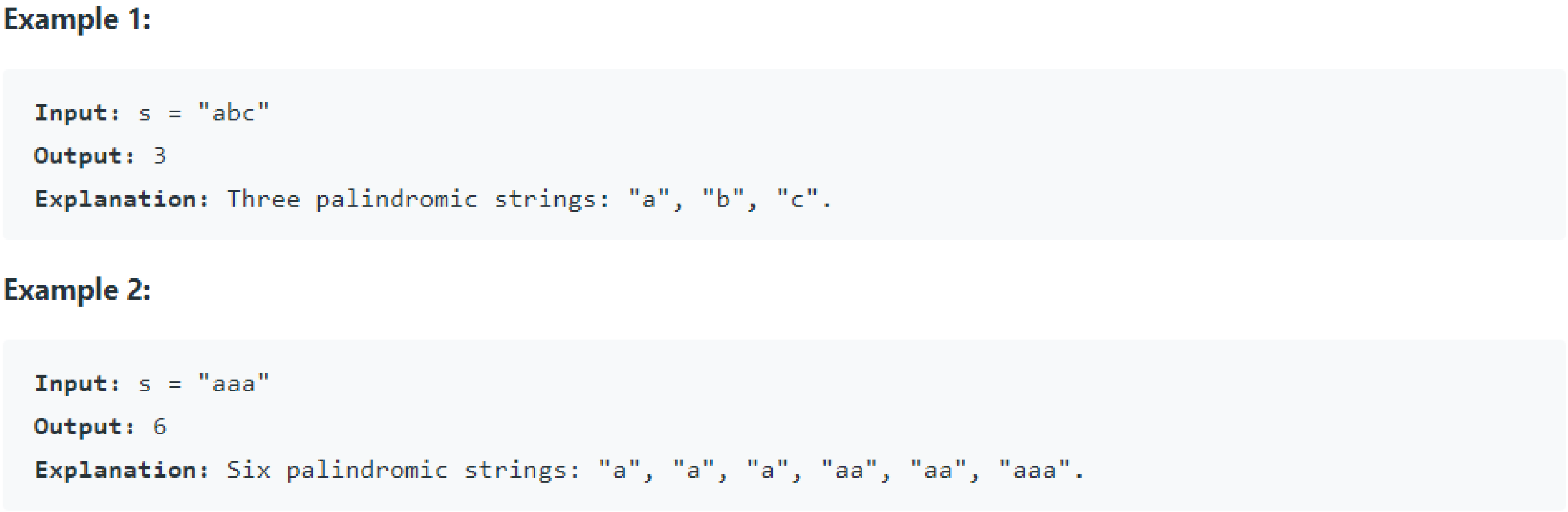


```
647 | Medium | Palindrome Substrings | Recursion/ DP

Given a string s, return the number of palindromic substrings in it.
A string is a palindrome when it reads the same backward as forward.

A substring is a contiguous sequence of characters within the string.

Constraints:
1 <= s.length <= 1000
s consists of lowercase English letters.
```

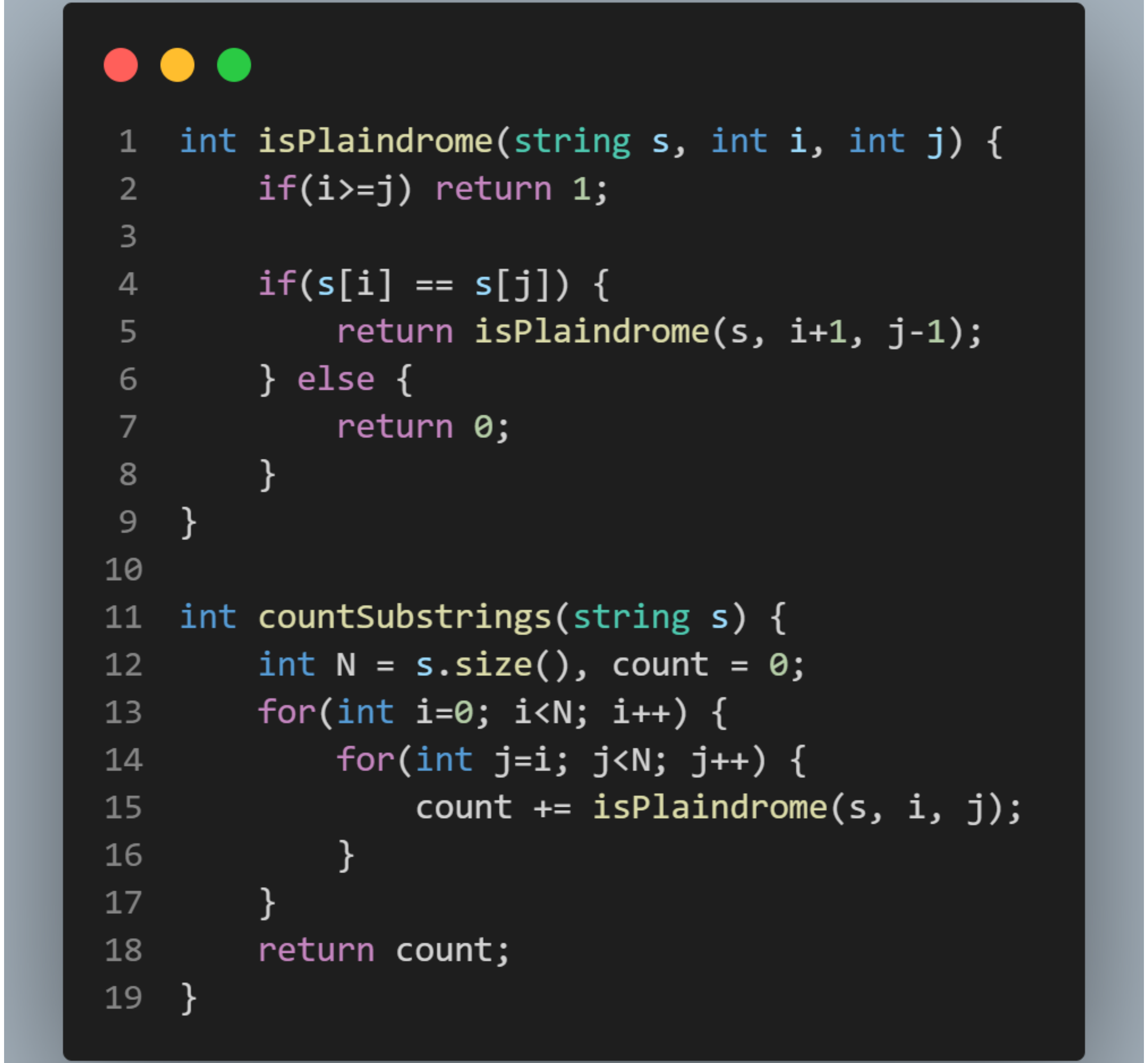


Example 1:

```
Input: s = "abc"
Output: 3
Explanation: Three palindromic strings: "a", "b", "c".
```

Example 2:

```
Input: s = "aaa"
Output: 6
Explanation: Six palindromic strings: "a", "a", "a", "aa", "aa", "aaa".
```

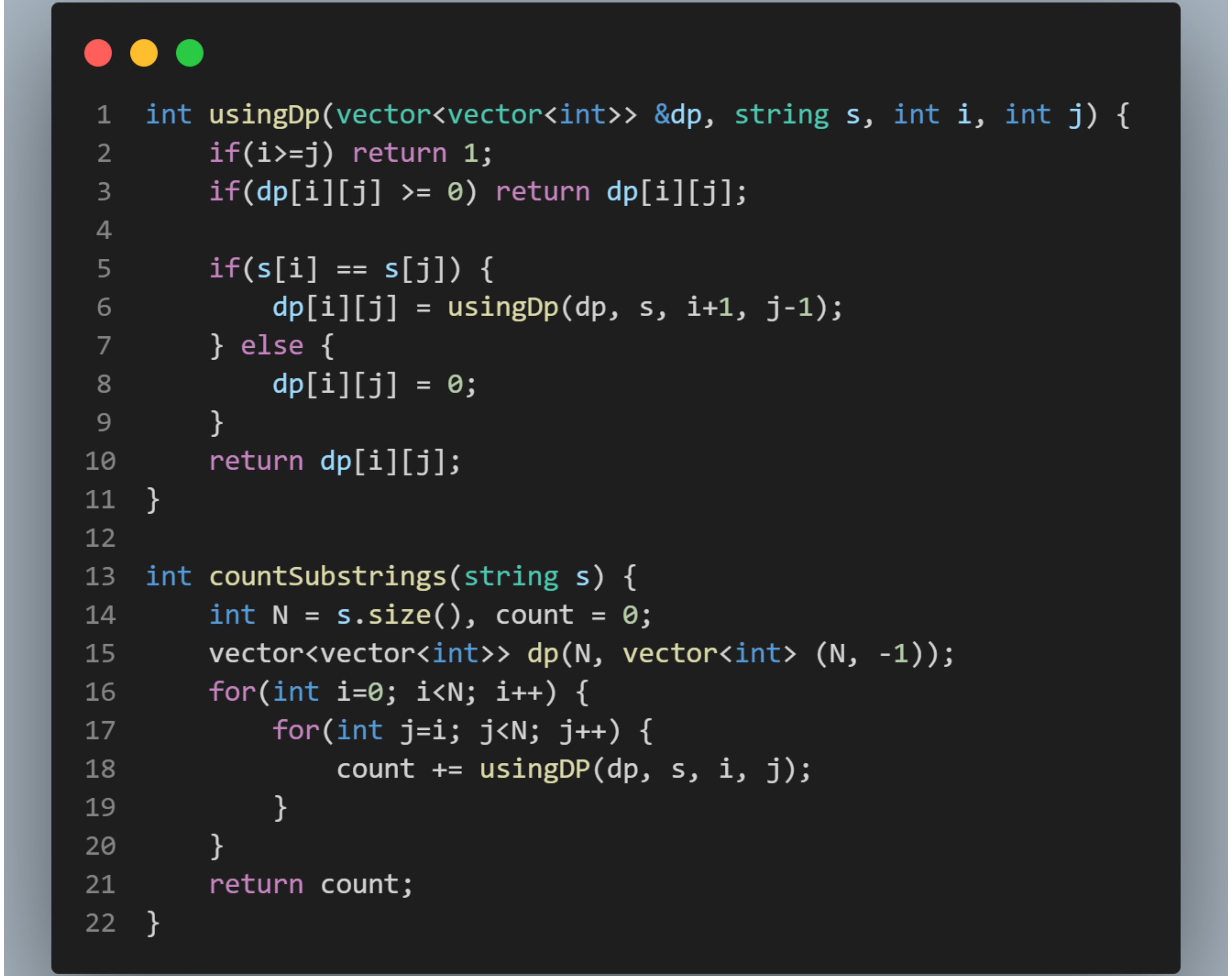


```cpp
int isPlaindrome(string s, int i, int j) {
    if(i>=j) return 1;

    if(s[i] == s[j]) {
        return isPlaindrome(s, i+1, j-1);
    } else {
        return 0;
    }
}

int countSubstrings(string s) {
    int N = s.size(), count = 0;
    for(int i=0; i<N; i++) {
        for(int j=i; j<N; j++) {
            count += isPlaindrome(s, i, j);
        }
    }
    return count;
}
```

In the approach, the time complexity is O(N^3) & Space is O(N). So, I got Time Limit Exceeded. Now, We can further optimise the time, with dynamic programming.



```
int usingDp(vector<vector<int>> &dp, string s, int i, int j) {
    if(i>=j) return 1;
    if(dp[i][j] >= 0) return dp[i][j];

    if(s[i] == s[j]) {
        dp[i][j] = usingDp(dp, s, i+1, j-1);
    } else {
        dp[i][j] = 0;
    }
    return dp[i][j];
}

int countSubstrings(string s) {
    int N = s.size(), count = 0;
    vector<vector<int>> dp(N, vector<int> (N, -1));
    for(int i=0; i<N; i++) {
        for(int j=i; j<N; j++) {
            count += usingDP(dp, s, i, j);
        }
    }
    return count;
}
```

#100daysofDSA

/rvislive

Rakesh Vishwakarma