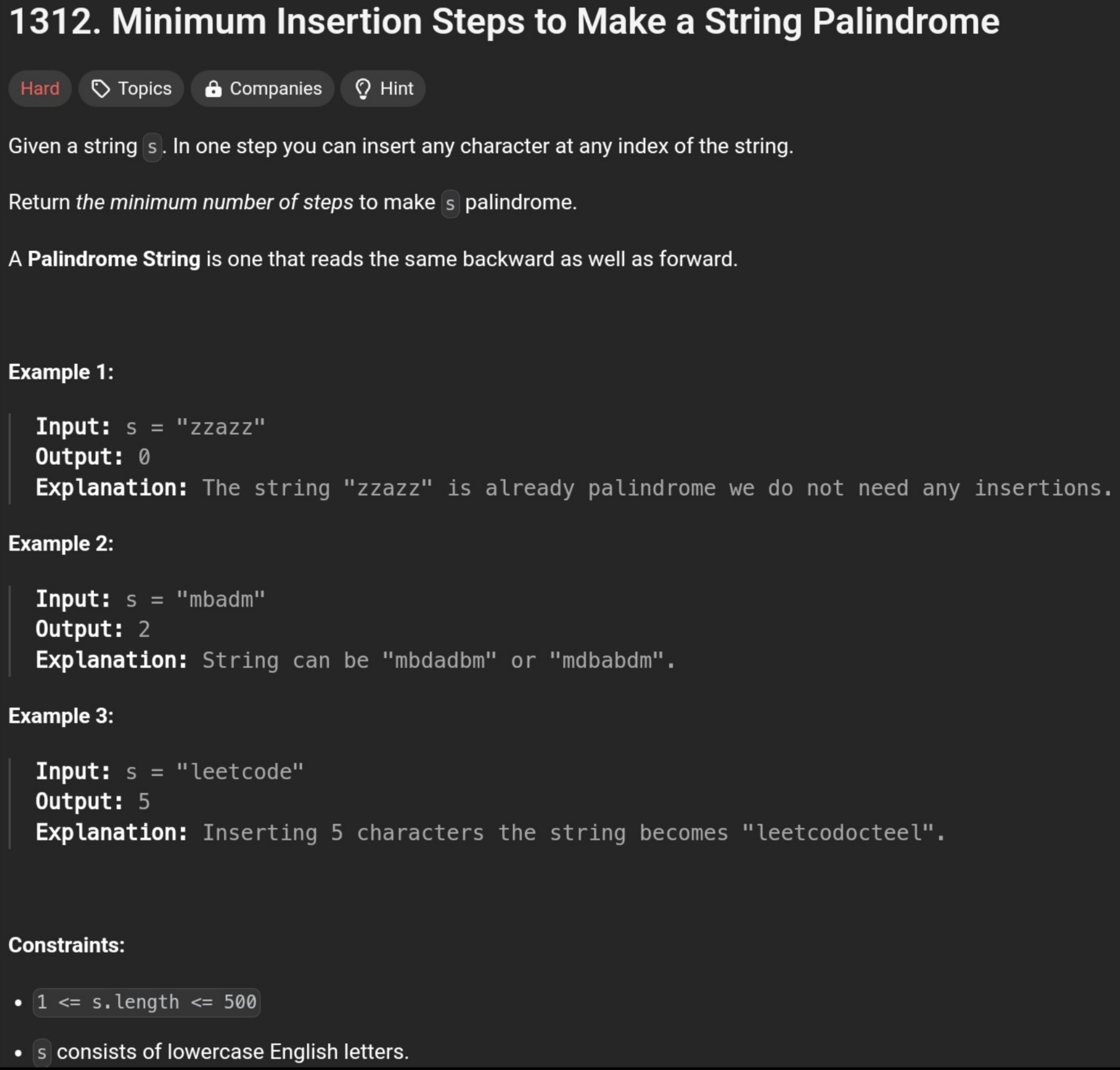


1312. Minimum Insertion Steps to Make a String Palindrome

Given a string `s`. In one step you can insert any character at any index of the string.

Return *the minimum number of steps* to make `s` palindrome.

A **Palindrome String** is one that reads the same backward as well as forward.

Example 1:

```
Input: s = "zzazz"
Output: 0
Explanation: The string "zzazz" is already palindrome we do not need any insertions.
```

Example 2:

```
Input: s = "mbadm"
Output: 2
Explanation: String can be "mbdadbm" or "mdbabdm".
```

Example 3:

```
Input: s = "leetcode"
Output: 5
Explanation: Inserting 5 characters the string becomes "leetcodocteel".
```

Constraints:

- `1 <= s.length <= 500`
- `s` consists of lowercase English letters.

Approach 1:

S = "leetcode"

The idea is, To get minimum insertions we should not disturb the already existing palindromic subsequence in the string.

So we keep the already existing longest palindromic subsequence intact.

Longest palindromic subsequence = "e e e"

Longest palindromic subsequence = e e e
Remaining characters = lt cod

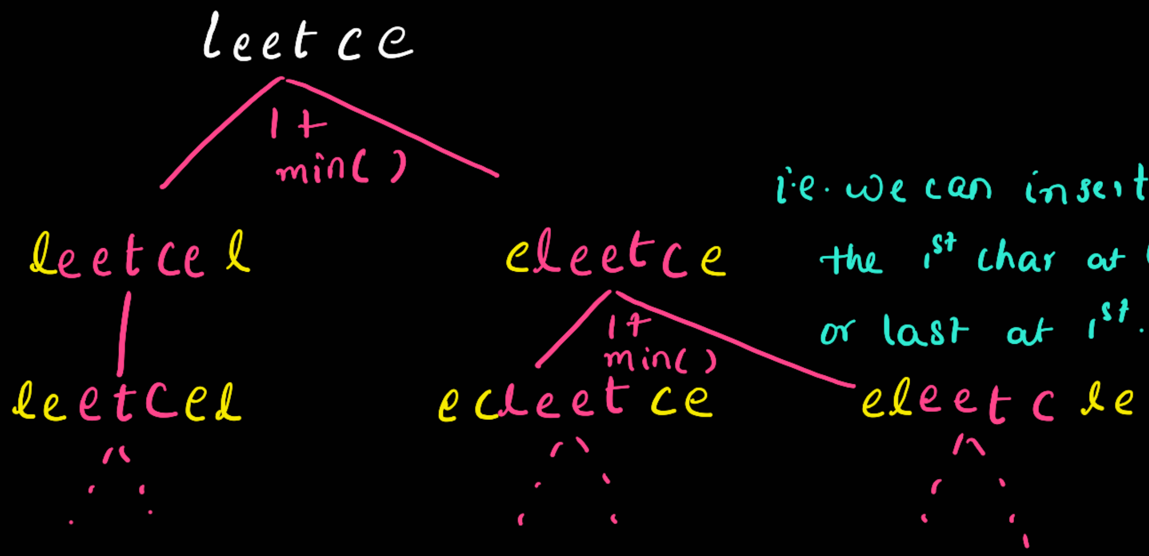
eltcodedoctle

Hence min insertions required = 5
i.e. n - longest palindromic subsequence

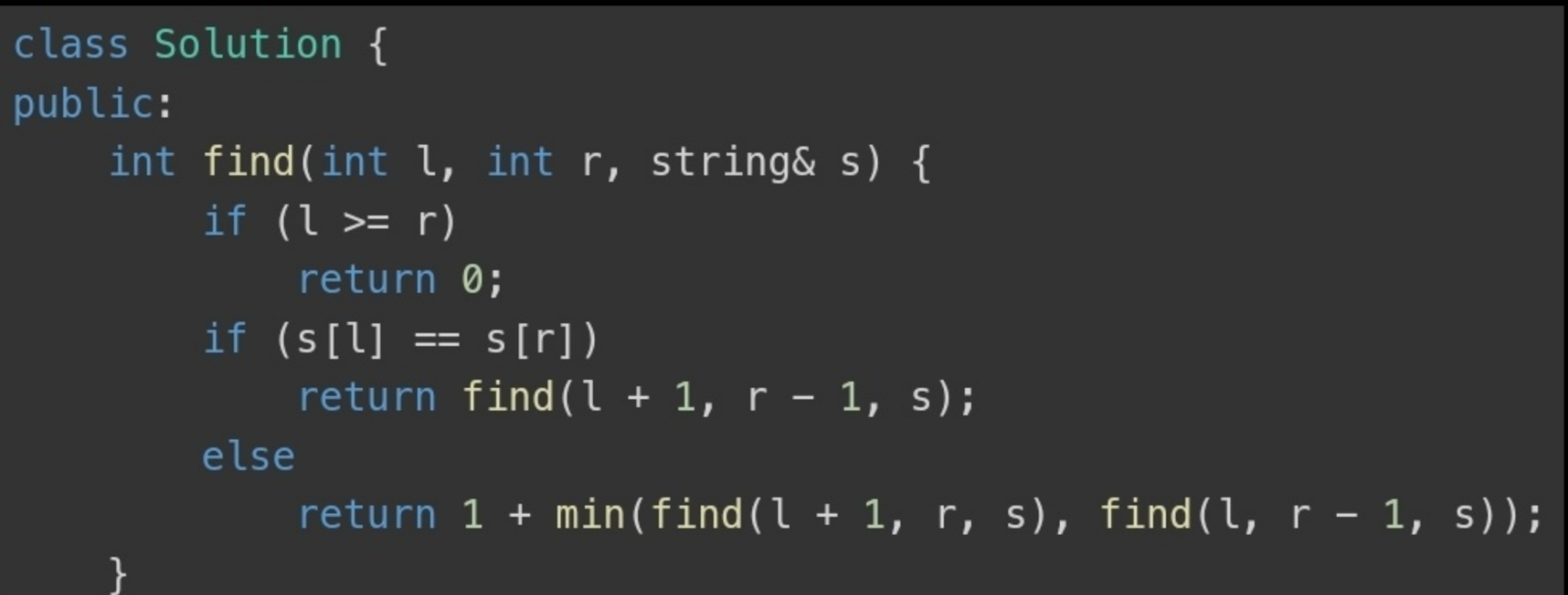
Now this problem got reduced to finding longest palindromic subsequence #516

T(n): T(n) of #516
S(n): S(n) of #516

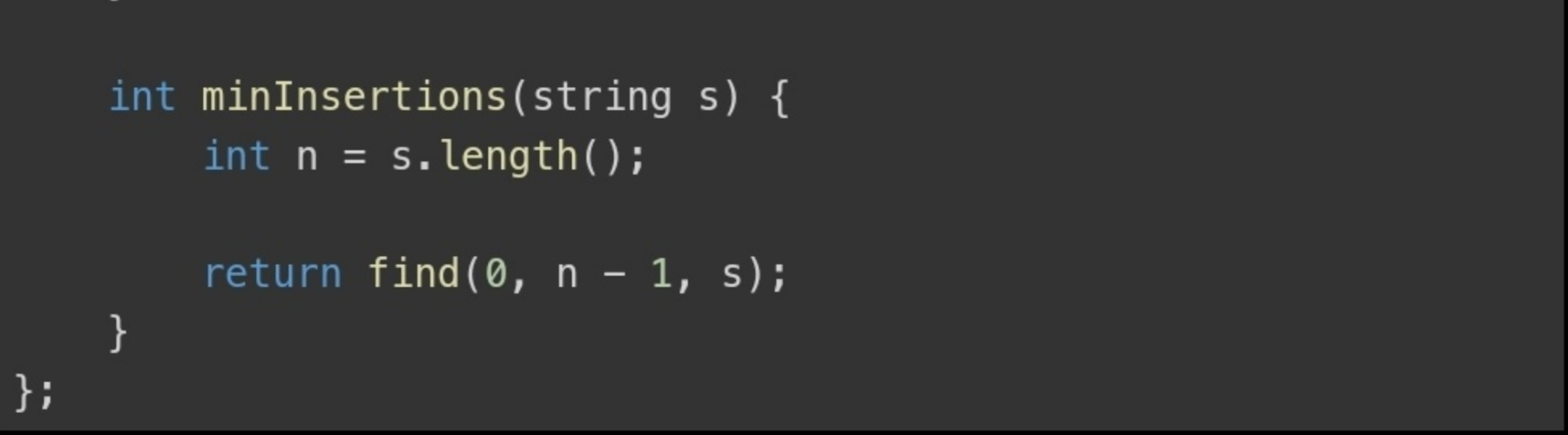
Approach 2: Recursion



i.e. we can insert the 1st char at last or last at 1st.



```
class Solution {
public:
    int find(int l, int r, string& s) {
        if (l >= r)
            return 0;
        if (s[l] == s[r])
            return find(l + 1, r - 1, s);
        else
            return 1 + min(find(l + 1, r, s), find(l, r - 1, s));
    }
```



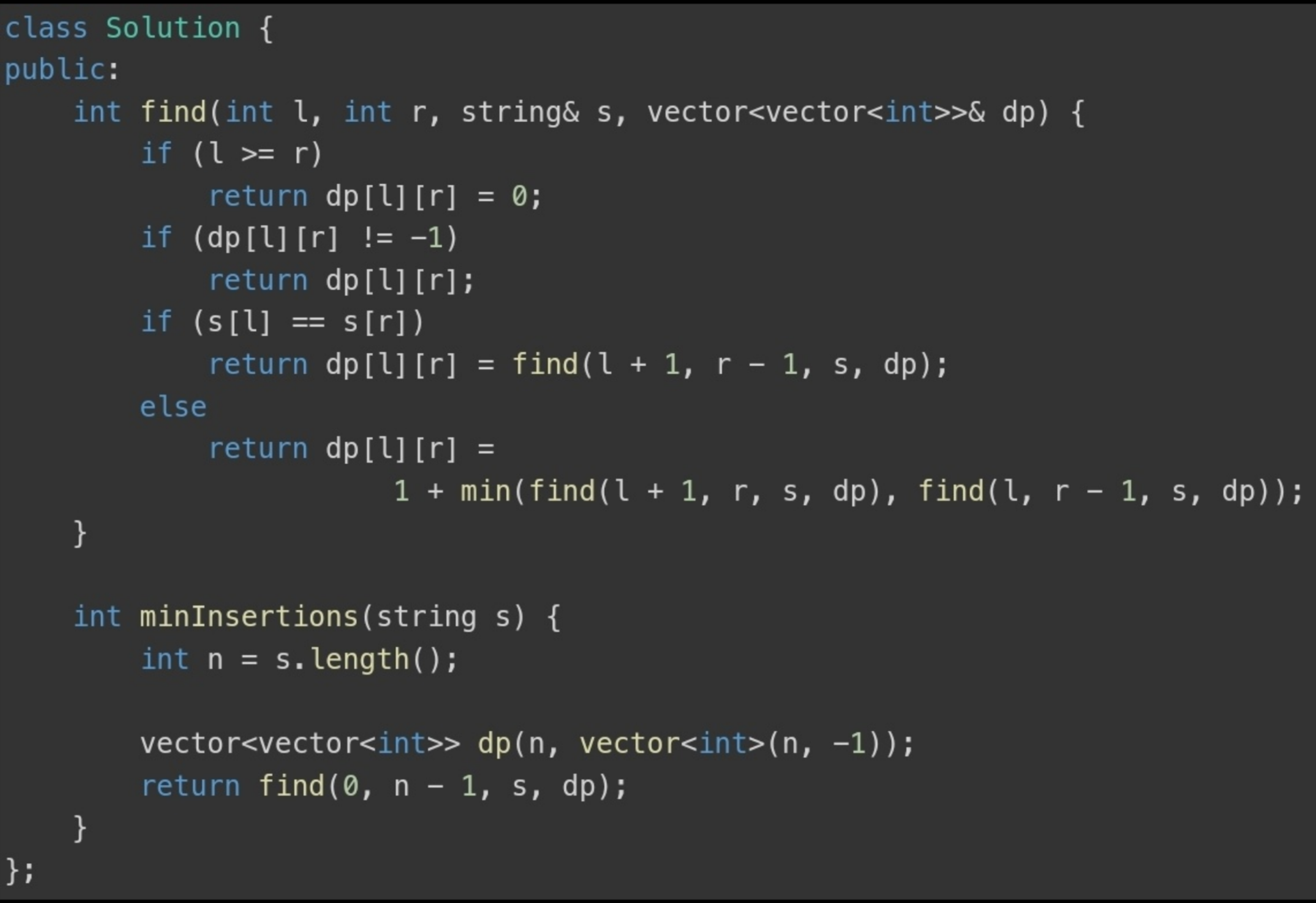
```
    int minInsertions(string s) {
        int n = s.length();

        return find(0, n - 1, s);
    }
};
```

$T(n) : O(2^n)$

$S(n) : O(n)$

Approach 3: Memoization

if we take larger strings, we can identify the overlapping subproblems.



```
class Solution {
public:
    int find(int l, int r, string& s, vector<vector<int>>& dp) {
        if (l >= r)
            return dp[l][r] = 0;
        if (dp[l][r] != -1)
            return dp[l][r];
        if (s[l] == s[r])
            return dp[l][r] = find(l + 1, r - 1, s, dp);
        else
            return dp[l][r] =
                       1 + min(find(l + 1, r, s, dp), find(l, r - 1, s, dp));
    }

    int minInsertions(string s) {
        int n = s.length();

        vector<vector<int>> dp(n, vector<int>(n, -1));
        return find(0, n - 1, s, dp);
    }
};
```

$T(n) : O(n^2)$

$S(n) : O(n^2) + O(n)$