132. Palindrome Partitioning II

Given a string `s`, partition `s` such that every substring of the partition is a palindrome.

Return *the minimum cuts needed* for a palindrome partitioning of `s`.

Example 1:

```
Input: s = "aab"
Output: 1
Explanation: The palindrome partitioning ["aa","b"] could be produced
using 1 cut.
```

Example 2:

```
Input: s = "a"
Output: 0
```

Example 3:

```
Input: s = "ab"
Output: 1
```

Constraints:

- `1 <= s.length <= 2000`
- `s` consists of lower-case English letters only. cg

The below method is O(N^3)

```
import sys
def palindromicPartition(s):
    dp = [[False] * len(s) for _ in range(len(s))]
    for gap in range(len(s)):
        i = 0
        j = gap
        while j < len(s):
            if gap == 0:
                dp[i][j] = True
            elif gap == 1:
                dp[i][j] = s[i] == s[j]
```

```
            else:
                if s[i] == s[j] and dp[i + 1][j - 1]:
                    dp[i][j] = True
            i = i+1
            j = j+1

    storage = [[0] * len(s) for _ in range(len(s))]
    for gap in range(len(s)):
        i = 0
        j = gap
        while j < len(s):
            if gap == 0:
                storage[i][j] = 0
            elif gap == 1:
                if s[i] == s[j]:
                    storage[i][j] = 0
                else:
                    storage[i][j] = 1
            else:
                if s[i] == s[j] and dp[i + 1][j - 1]:
                    storage[i][j] = 0
                else:
                    if dp[i][j]:
                        storage[i][j] = 0
                    else:
                        temp = sys.maxsize
                        for  k in range(i,j):
                            left = storage[i][k]
                            right = storage[k+1][j]
                            temp = min(temp,left+right+1)

                        storage[i][j] = temp
            i = i+1
            j = j+1


    return storage[0][-1]
```

Below is O(N^2) method

```
import sys
```

```python
def palindromicPartition(s):
    dp = [[False] * len(s) for _ in range(len(s))]
    for gap in range(len(s)):
        i = 0
        j = gap
        while j < len(s):
            if gap == 0:
                dp[i][j] = True
            elif gap == 1:
                dp[i][j] = s[i] == s[j]
            else:
                if s[i] == s[j] and dp[i + 1][j - 1]:
                    dp[i][j] = True
            i = i + 1
            j = j + 1

    storage = [0] * len(s)
    storage[0] = 0
    for j in range(1, len(s)):
        if dp[0][j]:
            storage[j] = 0
        else:
            temp = sys.maxsize
            for i in range(j, 0, -1):
                if dp[i][j]:
                    temp = min(temp, storage[i - 1])
            storage[j] = temp + 1

    return storage[-1]
    # print(storage)



s = "abccbc"
print(palindromicPartition(s))
```