回文子串

647. 回文子串

难度 中等 381

给定一个字符串，你的任务是计算这个字符串中有多少个回文子串。

具有不同开始位置或结束位置的子串，即使是由相同的字符组成，也会被视作不同的子串。

示例 1：

```
输入："abc"
输出：3
解释：三个回文子串："a", "b", "c"
```

示例 2：

```
输入："aaa"
输出：6
解释：6个回文子串："a", "a", "a", "aa", "aa", "aaa"
```

思路：

1 动态规划

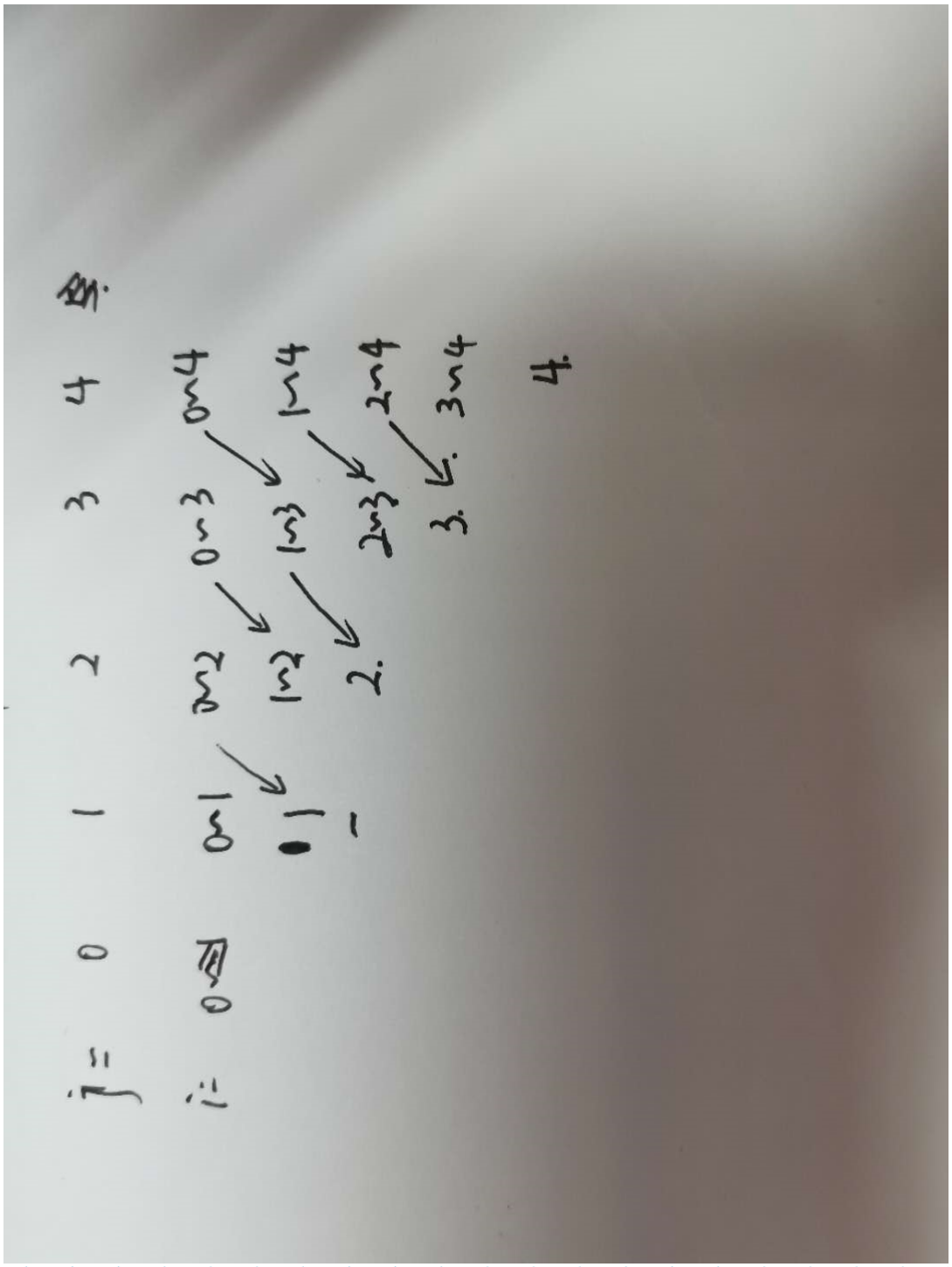
设置一个dp[i][j],， 表示字串[i,j]是否是一个回文串

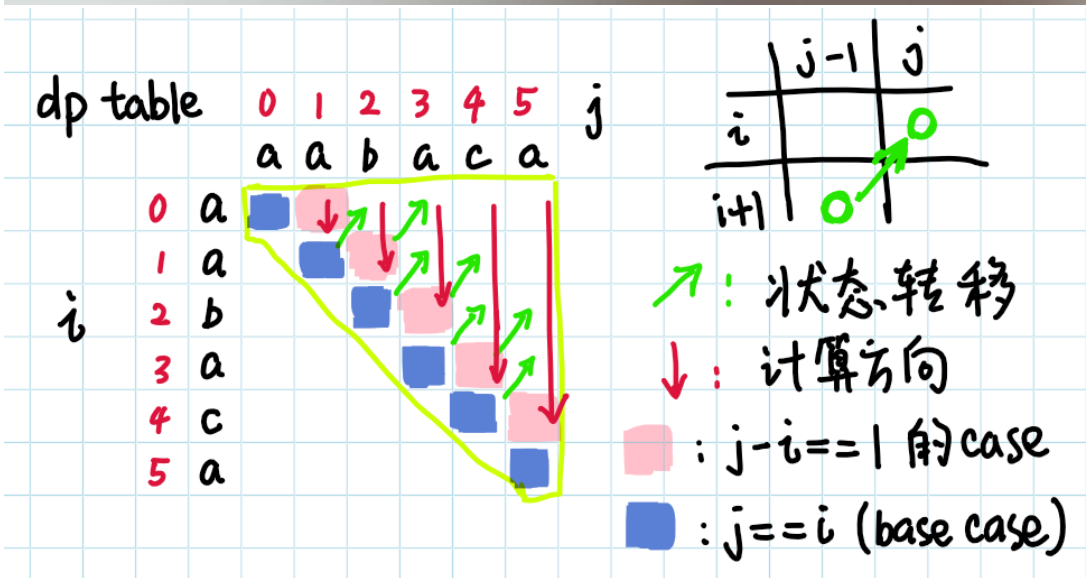
遍历所有可能的情况：

1 子串长度为1，则其是回文串

2 子串长度为2，如果两个字符相等，则是回文串

3 子串长度>2，如果首尾两个字符相等，中间的是回文串dp[i+1][j−1]，则是回文串






```python
def countSubstrings(self, s: str) -> int:
```

```
        size = len(s)
        dp = [[False] *size for _ in range(size)]
        count = 0
        for j in range(size):
            for i in range(0, j+1):
                %23 子串长度
                length = j -i + 1
                %23 只有一个字符，那么直接就是回文串
                if length == 1:
                    dp[i][j] = True
                    count += 1
                %23 只有两个字符，相等才是回文串
                if length == 2 and s[i] == s[j]:
                    dp[i][j] = True
                    count += 1
                %23 对于多个字符，首尾相同，且除去首尾的子串是回文串，
                if length > 2 and s[i] == s[j] and dp[i+1][j-1] is True:
                    dp[i][j] = True
                    count += 1
        return count
```

2 中心扩展

假设 $n = 4$，我们可以把可能的回文中心列出来：

编号 i	回文中心左起始位置 l_i	回文中心右起始位置 r_i
0	0	0
1	0	1
2	1	1
3	1	2
4	2	2
5	2	3
6	3	3

可以看出，回文中心有 2 *n −1 个，回文中心左起始位置是 i//2 , 回文中心又起始位置是 i//2 + i mod 2，依次遍历回文中心，从回文中心想两边扩展，如果对应的s[l] == s[r]，则把结果 ++

```
def countSubstrings(self, s: str) -> int:
        size = len(s)
        ans = 0
        for i in range(2*size -1):
            l,r = i//2, i//2 + i%2
            while l >=0 and r < size and s[l] == s[r]:
                l -= 1
                r += 1
                ans += 1
        return ans
```

时间复杂度O(n^2)

空间复杂度O(1)

Manacher 算法

```python
def countSubstrings(self, s: str) -> int:
        s = "%23" + "%23".join(s) + "%23"
        right = -1
        maxlen = [0]*len(s)
        res = 0
        for i in range(len(s)):
            if i<=right:
                im = 2*j-i
                minlen = min(maxlen[im],right-i)
                tmp = self.expand(s,i-minlen,i+minlen)
            else:
                tmp = self.expand(s,i,i)
            maxlen[i] = tmp
            if i+tmp>right:
                right = i+tmp
                j = i
            if i%2==1:
                res += maxlen[i]//2 + 1
            else:
                res += (maxlen[i]+1)//2
        return res

    def expand(self,s,l,r):
        while 0<=l and r<len(s) and s[l] == s[r]:
            l -= 1
            r += 1
        return (r-l)//2 - 1
```

时间复杂度 O(n)

空间复杂度 O(1)