

力扣 664. 奇怪的打印机

题目描述

有台奇怪的打印机有以下两个特殊要求：

打印机每次只能打印由 **同一个字符** 组成的序列。

每次可以在任意起始和结束位置打印新字符，并且会覆盖掉原来已有的字符。

给你一个字符串 s ，你的任务是计算这个打印机打印它需要的最少打印次数。

示例 1：

输入： $s = "aaabbb"$ 输出：2 解释：首先打印 "aaa" 然后打印 "bbb"。

示例 2：

输入： $s = "aba"$ 输出：2 解释：首先打印 "aaa" 然后在第二个位置打印 "b" 覆盖掉原来的字符 'a'。

提示：

- $1 \leq s.length \leq 100$
- s 由小写英文字母组成

解决方案：动态规划

我们可以使用 **动态规划** 解决本题

一、状态表示

令 $f[i][j]$ 表示打印完成区间 $[i, j]$ 的 **最少操作数**

状态设计原则

- 状态设计是不是最终能包含答案
- 通过简单枚举可以找出正确答案

二、状态转移

当我们尝试计算出 $f[i][j]$ 时，需要考虑两种情况：

- $s[i] = s[j]$ 即 **区间两端的字符相同时**，那么当我们打印左侧字符 $s[i]$ 时，可以顺便打印右侧字符 $s[j]$ ，这样我们即可忽略右侧字符对该区间的影响，只需要考虑如何尽快打印完区间 $[i, j - 1]$ 即可，即此时有

$$f[i][j] = f[i][j - 1]$$

- $s[i] \neq s[j]$ 即区间两端的字符不同，那么我们 **需要分别完成该区间的左右两部分的打印**。我们记两部分分别为区间 $[i, k]$ 和区间 $[k + 1, j]$ (其中 $i \leq k < j$)，此时

$$f[i][j] = \min_{k=i}^{j-1} (f[i][k] + f[k+1][j])$$

注意：这里 k 的取值范围值得仔细思考，因为是划分成两段，设第一段结束点为 k ，第二段的开始点为 $k+1$ ，则有 $k \geq i, k+1 \leq j$ ，即： $i \leq k < j$

$i = k$ 时是成立的，表示只有 i 点打印一次，从 $k+1$ 开始至 j 打印其它的(不一定是同一种噢~)

总结状态转移方程为：

```
\large \displaystyle f[i][j] =
\left(\begin{matrix}
f[i][j-1] \ \& \ s[i]=s[j] \ \\
\displaystyle \min_{k=i}^{j-1} \{f[i][k]+f[k+1][j] \ \& \ s[i] \neq s[j] \}
\end{matrix}\right)
```

三、边界与答案

- 边界条件为 $f[i][i] = 1$ ，对于长度为 1 的区间，最少打印 1 次
- 最后的答案为 $f[0][n-1]$

四、枚举顺序

注意到 $f[i][j]$ 的计算需要用到 $f[i][k]$ 和 $f[k+1][j]$

$$i \leq k < j$$

为了保证动态规划的计算过程满足无后效性，在实际代码中，我们需要改变动态规划的计算顺序，从大到小地枚举 i ，并从小到大地枚举 j ，这样可以保证当计算 $f[i][j]$ 时， $f[i][k]$ 和 $f[k+1][j]$ 都已经被计算过。

五、复杂度分析

时间复杂度： $O(n^3)$ ，其中 n 是字符串的长度。空间复杂度： $O(n^2)$ ，其中 n 是字符串的长度。我们需要保存所有 n^2 个状态。

六、动态规划代码

```
#include <bits/stdc++.h>

using namespace std;
const int N = 110;
/*
https://leetcode.cn/problems/strange-printer/

输入：s = "aaabbb"
输出：2
解释：首先打印 "aaa" 然后打印 "bbb"。
示例 2：
```

输入: s = "aba"

输出: 2

解释: 首先打印 "aaa" 然后在第二个位置打印 "b" 覆盖掉原来的字符 'a'。

```

*/
int f[N][N];

int main() {
    string s;

    cin >> s;

    int n = s.size();
    memset(f, 0x3f, sizeof f);
    for (int i = n - 1; i >= 0; i--) {
        f[i][i] = 1;

        for (int j = i + 1; j < n; j++) {
            if (s[i] == s[j])
                f[i][j] = f[i][j - 1];
            else {
                for (int k = i; k < j; k++)
                    f[i][j] = min(f[i][j], f[i][k] + f[k + 1][j]);
            }
        }
    }
    cout << f[0][n - 1] << endl;
    return 0;
}

```

七、记忆化搜索代码

```

#include <bits/stdc++.h>

using namespace std;
const int N = 110;
const int INF = 0x3f3f3f3f;
string s;
int f[N][N];

int dfs(int l, int r) {
    if (l == r) return 1;
    if (f[l][r]) return f[l][r];
    int res = INF;

    if (s[l] == s[r]) return dfs(l, r - 1);

    for (int i = l; i < r; i++)
        res = min(res, dfs(l, i) + dfs(i + 1, r));

    return f[l][r] = res;
}

```

```
}

int main() {
    cin >> s;
    cout << dfs(0, s.size() - 1) << endl;
    return 0;
}
```