算法分析与设计E5-B

22373340 詹佳博

题目描述

Bellalabella 今天学习了回文串的内容，一个字符串是回文串当且仅当将其翻转后和原串相同。

Bellalabella 想知道对于一个字符串 S，它的最长回文子串长度是多少。

输入格式

第一行一个正整数 $t(1 \leq t \leq 10)$，表示数据组数。

对于每组数据，一行一个由小写字母组成的字符串S，含义同题目描述。

保证对于所有数据 $|S|$的和不超过 2×10^3。

输出格式

对于每组数据，输出一行一个正整数，表示串S的最长回文子串长度。

题目分析

（BF遍历寻找）从头开始寻找找到所有子串，判定是否是回文串，保存最长长度的回文串。

（动态规划）用一个二维数组dp[i][j]表示字符串的第i个到第j个是否是回文串。然后来找出最大的j-i即可。

题目求解

- BF算法

```
#include <bits/stdc++.h>
using namespace std;

string str;

bool pali(string str, int i, int j) {
    while (i < j) {
        if (str[i] != str[j]) {
            return false;
        }
        i++;
        j--;
    }
    return true;
}
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    int t;
    cin >> t;
    while (t--) {
```

```
        cin >> str;
        int len = str.length();
        if (len < 2) {
            cout << 1 << endl;
        } else {
            int maxlen = 0;
            for (int i = 0; i < len - 1; i++) {
                for (int j = i + 1; j < len; j++) {
                    if (j - i + 1 > maxlen && pali(str, i, j)) {
                        maxlen = j - i + 1;
                    }
                }
            }
            cout << maxlen << endl;
        }
    }
}
```

- dp算法

```
#include <bits/stdc++.h>
using namespace std;

string str;
bool dp[2020][2020];
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    int t;
    cin >> t;
    while (t--) {
        memset(dp, 0, sizeof dp);
        for (int i = 0; i < 2020; i++) {
            dp[i][i] = 1;
        }
        cin >> str;
        int len = str.length();
        int maxlen = 0;
        for (int j = 1; j < len; j++) {
            for (int i = 0; i < j; i++) {
                if (str[i] == str[j] && j != i + 1) {
                    dp[i][j] = dp[i + 1][j - 1];
                } else if (str[i] == str[j] && j == i + 1) {
                    dp[i][j] = 1;
                } else {
                    dp[i][j] = 0;
                }
                if (dp[i][j] && j - i + 1 > maxlen) {
                    maxlen = j - i + 1;
                }
            }
        }
        cout << maxlen << endl;
    }
```

```
}
```

时间复杂度

经分析，BF算法$O(n^3)$，dp算法$O(n^2)$。