# 250704小码哥的英语 （题解）

> 题目链接：
> http://www.matiji.net/exam/brushquestion/6/4446/16A92C42378232DEB56179D9C70DC45C?from=1

小码哥在做英语单选题的时候突然想到了一个很有意思的问题：单选题的答案都是A或者B,那么很多道题目的答案组成的就是一个A和B组成的字符串。如果两个相邻的题目的答案是相同的，小码哥就可以修改这两个题目的答案，改成都是A、都是B或者一个是A另外一个是B,小码哥把这个定义为一次操作。那么对于任意一个答案字符串，最少需要多少次操作，才能使得它满足：任意相邻的题目的答案都不同。

格式
输入格式：
第一行包含一个数字n(1≤n≤1000)，表示字符串的个数；后面的n行，每一行表示一个长度为L(1≤工≤10000)的，由字符A和B随机组成的字符串。

输出格式：
输出有多行，每一行表示对应字符串需要的最少的操作次数T(0≤T≤10000)。

样例1
输入：2
AAABAB
AABBBAB
输出：1
2

---

# 解法一：

题目本质：

把字符串变成：
ABABAB...
或
BABABA...

只有这两种情况

对于每个位置每个位置：目标字符不符 ⇒ 需修改

统计两种情况的最小修改次数

示例

输入：
AAABAB

1. 目标：ABABAB
原串：  A A A B A B
目标：  A B A B A B
变化：  0 1 0 0 0 0 ⇒ 1次操作

2. 目标：BABABA
原串：  A A A B A B
目标：  B A B A B A
变化：  1 0 1 1 1 1 ⇒ 5次操作

所以最优 = min ( 1 , 5 ) = 1 次

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

int main() {
    int n;
    cin >> n;
    while (n--) {
        string s;
        cin >> s;
        int cnt1 = 0, cnt2 = 0;
        for (int i = 0; i < s.size(); ++i) {
            char expect1 = (i % 2 == 0) ? 'A' : 'B';
            char expect2 = (i % 2 == 0) ? 'B' : 'A';
            if (s[i] != expect1) cnt1++;
            if (s[i] != expect2) cnt2++;
        }
        cout << min(cnt1, cnt2) << "\n";
    }
    return 0;
}
```

---

## 动态规划：

```cpp
#完整代码
#include <bits/stdc++.h>
using namespace std;

int main() {
```

```
    int n;
    cin >> n;
    while (n--) {
        string s;
        cin >> s;
        int len = s.size();
        // dp[i][c]: 到第 i 位字符选 c，最少操作次数
        vector<vector<int>> dp(len, vector<int>(2, INT_MAX));

        dp[0][0] = (s[0] != 'A');
        dp[0][1] = (s[0] != 'B');

        for (int i = 1; i < len; ++i) {
            // 如果当前位置选A（c=0），上一位置必须是B（1）
            dp[i][0] = dp[i-1][1] + (s[i] != 'A');
            // 如果当前位置选B（c=1），上一位置必须是A（0）
            dp[i][1] = dp[i-1][0] + (s[i] != 'B');
        }

        cout << min(dp[len-1][0], dp[len-1][1]) << "\n";
    }
    return 0;
}
```

## 动态规划思路

### 状态设计

设：

```
dp[i][0] = 前 i 位，当前第 i 位是 'A'，最小修改次数
dp[i][1] = 前 i 位，当前第 i 位是 'B'，最小修改次数
```

注意：

- 状态表示的是前缀 `[0 ~ i]` 的最优结果
- `dp[i][0]` 与 `dp[i][1]` 代表两种互斥选择路径

### 初始状态

第 0 位：

```
dp[0][0] = (s[0] != 'A'); // 若本来是 'A'，不需操作，否则改成 'A'
dp[0][1] = (s[0] != 'B'); // 同理
```

## 状态转移

对于第 `i` 位：

1. 若当前为 `'A'`：
   - 上一位必须为 `'B'`，否则相邻冲突
   - 方案：

```
dp[i][0] = dp[i - 1][1] + (s[i] != 'A');
```

2. 若当前为 `'B'`：
   - 上一位必须为 `'A'`
   - 方案：

```
dp[i][1] = dp[i - 1][0] + (s[i] != 'B');
```

**解释：**

- `(s[i] != 'A')`：若当前位置不是期望字符，需操作一次
- 递推保证严格交替，避免冲突

## 最终答案

字符串最后一位可以是：

- `'A'`：`dp[n - 1][0]`
- `'B'`：`dp[n - 1][1]`

取两种方案的最小值：

```
min(dp[n - 1][0], dp[n - 1][1])
```

## 思路总结

```
dp[0][0] = 是否改成A
dp[0][1] = 是否改成B

for i = 1 到 n-1:
    dp[i][0] = dp[i - 1][1] + (s[i] != 'A')  // 上一位B，这位A
    dp[i][1] = dp[i - 1][0] + (s[i] != 'B')  // 上一位A，这位B
```

```
答案 = min(dp[n - 1][0], dp[n - 1][1])
```