S193 - Alice’s Housekeeper

Author: Percy Wong
Speaker: Susanna Chan

香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

Story of S193

Because of the coming exam period, Alice feels very stressful and gets mad easily. When she is angry, she will scold at her housekeepers, even worse she may punish them!

You, as one of the housekeepers of Alice, do not want to get punished. One way to keep your master in a good mood is to serve her favorite dishes as dinner. You have got a list of K dishes Alice likes, denoted as ...

Story of S193

~~Because of the coming exam period, Alice feels very stressful and gets mad easily. When she is angry, she will scold at her housekeepers, even worse she may punish them!~~

~~You, as one of the housekeepers of Alice, do not want to get punished. One way to keep your master in a good mood is to serve her favorite dishes as dinner. You have got a list of K dishes Alice likes, denoted as ...~~

Too long didn’t read :P

Let’s make it simple...

Given a string *s*... (*In the following slides, characters of *s* are 1-indexed*)

Output a string (length = *N*) derived from *s* that fulfills the following requirements:

1. No adjacent characters are equal
2. *s*[1] != *s*[*N*] (since the menu is cyclic)
3. Consists of only the first K lowercase letters
4. Changes the least number of characters in *s*

If it is impossible to fulfill requirements, output “Impossible”

Subtasks

For all cases:
$1 \le N \le 1000000$, $2 \le K \le 26$

Subtask 1 (8 points): $N = 2$

Subtask 2 (21 points): $K = 2$

Subtask 3 (39 points): $1 \le N \le 5000$

Subtask 4 (32 points): no additional constraints

Statistics

Attempts: 81

Mean: 23.271

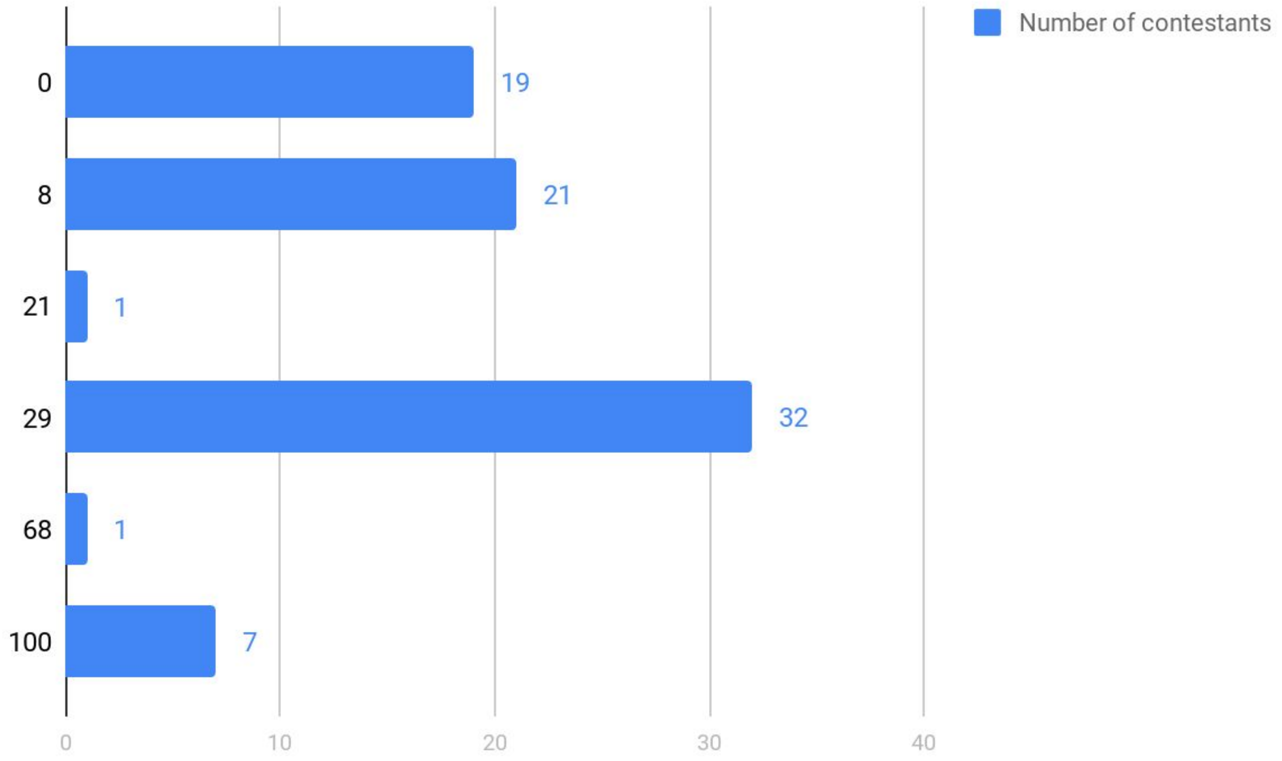
Standard Deviation: 27.261

First solved by **dbsgame** (1h 2m)

Statistics

Score Distribution

(among attempted contestants)



Subtask 1: N = 2 (8 points)

The strings can be classified into either one of the following 2 cases

Case 1: *s*[1] = *s*[2] → Change either *s*[1] or *s*[2] to other characters

```
IF s[1] = ‘a’ → Change s[1] to ‘b’ (final s = “ba”)       // initial s = “aa”
ELSE → Change s[1] to ‘a’ (s = “a#”)                      //(initial s = “##”)
    // ‘#’ = any lowercase letters other than ‘a’
```

Case 2: *s*[1] != *s*[2] → Output the original string

For data in this subtask, possible answers always exist.

Subtask 2: K = 2 (21 points)

There are only ‘a’s and ‘b’s in *s*

Key observations

1. There are only 2 possible arrangements
   - “ababab…” and “bababa…”
   - You can’t write ‘correct’ arrangements other than these two :o)
2. When *N* is an odd number, it is impossible to construct a string that satisfies the requirements
   - For arrangements above, *s*[1] is always equal to *s*[N] when *K* = 2 and *N* is odd
   - Example: “**a**b**a**” and “**b**a**b**”

Subtask 2: K = 2 (21 points)

Based on the observations...

```
IF $N \% 2 = 1$
    Output “Impossible”
ELSE
    Construct 2 strings $A$ = “ababab....” and $B$ = “bababa...” (length = N)
    Count the number of different characters between the $A$ and $s$ + $B$ and $s$
    Output the one with smaller difference ($A$ / $B$)
```

Time complexity: **O(*N*)**

Subtask 2… Special Case!

In fact, data with $K = 2$ are special cases in this question that requires special handling.

There is a general solution for the other cases :D

Minimizing the number of changes

We are only concerned about adjacent characters.

Hence for every character, we only need to check its previous and next characters. We can perform a sequential iteration to check the characters, so we can just compare the current character with the next character.

To minimize the number of changes, it’s obvious that we don’t need to modify adjacent characters when they are different. (i.e. $s[i] \mathrel{!}= s[i + 1]$). While for identical ones ($s[i] = s[i + 1]$), we need to perform some changes to them.

Minimizing the number of changes

You may have this question in mind: when we encounter *s*[*i*] = *s*[*i* + 1], which character should we modify? *s*[*i*] or *s*[*i* + 1]?

Observe how we change a subarray (continuous array) of consecutive characters.

Example 1: “aaaa” (length = 4 → even number)

If we change *s*[*i*], resulting string = “**b**a**b**a” [changes = 2]

If we change *s*[*i* + 1], resulting string = “a**b**a**b**” [changes = 2]

Hmm... we cannot make a conclusion...

Minimizing the number of changes

Example 2: “ddddd” (length = 5 → odd number)

If we change *s*[*i*], the resulting string = “**a**d**a**d**a**” [changes = 3]

If we change *s*[*i* + 1], the resulting string = “d**a**d**a**d” [changes = 2] :D

* Given that a complete subarray (continuous array) of identical characters is obtained, when there is *s*[*i*] = *s*[*i* + 1], we should always change *s*[*i* + 1] *

Notice that the next character for *s*[*N*] is *s*[1] :)

A general expression for the next character is *s*[*i* MOD *N* + 1] (1-indexed) or *s*[(*i* + 1) MOD *N*] (0-indexed).

Subtask 3: 1 ≤ N ≤ 5000 (8 + 21 + 39 = 68 points)

Now you know you should change the latter character when encountering identical characters.

But... HOW?

Subtask 3… Naive solution (WRONG)

For example, *s* = “aabcdefaaa”

If you naively iterate the string from index 1 to *N* and modify them as mentioned, you might obtain “**bc**bcdefa**b**a”… [changes = 3]

Though you can construct a string with no identical and adjacent characters, the number of changes is NOT optimal!

What about “**b**abcdefa**b**a”? [changes = 2]

Subtask 3: 1 ≤ N ≤ 5000 (8 + 21 + 39 = 68 points)

* Notice we need the **COMPLETE** subarray to make the aforementioned method correct. *

If the menu is not cyclic, we can iterate from index 1 of the string. Then we must encounter all the COMPLETE subarrays with identical characters.

Unfortunately, this menu IS cyclic!

Where is the starting position of the subarray?

Subtask 3: 1 ≤ N ≤ 5000 (8 + 21 + 39 = 68 points)

N ≤ 5000... a O(*N*^2) solution can pass the time limit :D

We know that checking and modifying takes O(*N*) time complexity, so adding an additional nested for-loop (loop *N* times) is still okay.

Recall the problem we are now facing is uncertain starting position of a continuous array of identical characters.

What we can do is exhaust every character as starting position, do modifications accordingly and check if the solution is optimal!

Subtask 3: 1 ≤ N ≤ 5000 (8 + 21 + 39 = 68 points)

```
Iterate every position in $s$ (denote the position = $j$)
    Iterate the whole string starting from index $j$ (current string position = $i$)
        Check if changes need to be done
        If changes needed → change $s[i + 1]$ such that
            $s[i] != s[i + 1]$ and $s[i + 1] != s[i + 2]$
        Increment number of changes
    Check if the new string is an optimal solution
```

Time complexity: **O(N^2)**

Subtask 3: 1 ≤ N ≤ 5000 (8 + 21 + 39 = 68 points)

Turns out when *K* != 2, possible answers almost always exist.

Except when *N* = 1…

If *N* = 1, the output is always “Impossible”!

A corner case can cost you many points :(

Subtask 4: No additional constraints (100 points)

If you have passed subtask 3, you are very close to full score!

Task: cut down the time complexity of your code → 10^6... a O(*N*) solution?

In subtask 3, you tried every position as the starting point.

Indeed, it is unnecessary to try most of the positions. WHY?

Subtask 4: No additional constraints (100 points)

There may be many subarray of identical characters, but the only one actually matters is the one connecting index 1 and index *N* (For example “**aa**bbb**aaa**”, the bolded subarray)

If we start to iterate the whole string starting from the first character of this subarray, or the first character after this subarray, we can actually encounter every complete subarray of continuous characters.

Subtask 4: No additional constraints (100 points)

Find the first character (position = *x*) such that *s*[*x*] != *s*[*N*].

Iterate the string starting from *s*[*x*].

When *s*[*i*] = *s*[*i* + 1], change the character *s*[*i* + 1] such that *s*[*i* + 1] != *s*[*i*] and *s*[*i* + 1] != *s*[*i* + 2] (e.g. “a**a**c” → “a**b**c”)

Time complexity: **O(*N*)**

Accepted