MY (EXTRA) ORDINARY LIFE (HTTP://EN.SHAFAETSPLANET.COM/)

TECH, TRAVEL AND EVERYTHING

HOME (HTTP://EN.SHAFAETSPLANET.COM/) / ABOUT (HTTP://EN.SHAFAETSPLANET.COM/SAMPLE-PAGE/)

PROBLEM SOLVING: COLORFUL BALLS (ICPC PRELI DHAKA 2018)

OCTOBER 8, 2018 (HTTP://EN.SHAFAETSPLANET.COM/PROBLEM-SOLVING-COLORFUL-BALLS-ICPC-PRELI-DHAKA-2018/) / LEAVE A COMMENT (HTTP://EN.SHAFAETSPLANET.COM/PROBLEM-SOLVING-COLORFUL-BALLS-ICPC-PRELI-DHAKA-2018/#RESPOND)

This problem appeared in the 2018 ACM ICPC Dhaka Regional (Preliminary round). This turned out to be quiet easy as more then 100 teams were able to solve it. Lets read the statement first:

Search … SEARCH

RECENT POSTS

Problem Solving: Colorful Balls (ICPC Preli Dhaka 2018) (http://en.shafaetsplanet.com/problem-solving-colorful-balls-icpc-preli-dhaka-2018/)

Intro to Staircase Nim (http://en.shafaetsplanet.com/staircase-nim/)

ARCHIVES

October 2018 (http://en.shafaetsplanet.com/2018/10/)

May 2018 (http://en.shafaetsplanet.com/2018/05/)

Alice has **N** balls arranged in a single line. The balls are either red(**R**), blue(**B**), green(**G**) or white(**W**). They can be represented by a string **S**. Every character of the string is either **R, B, G or W.**

In the beginning, also there are no two balls of the same color side by side (except white ball). For example GGWWB is not a valid string because there are two green balls together. But GWWB is a valid string as there are no two balls of same color side by side except white balls.

Alice needs to paint all the white balls in one of the other three colors in a way that there are no two balls of the same color side by side.

How many ways Alice can paint the balls? Print the solution modulo **1000000007 (10^9 + 7).**

Input

The first line contains the number of test cases **T (1≤T≤1000).** In each line of the test cases, there will be a string **S** of length **N (1≤N ≤100000).**

The total number of character in the input file will be less than **5 * 10^6.**

Output

For each test case, print the case number and the answer to the problem.

Sample Input

```
2
WWG
GWGWB
```

Sample Output

```
Case 1: 4
Case 2: 2
```

Time Limit: 1 seconds

```
Memory Limit: 64MB
```

The input size and the time limit suggests that this need be solved in $O(n)$ or better! You need some basic combinatorics and dynamic programming knowledge to solve this. Don't know dynamic programming? Don't worry, I hope after the end of the editorial you will have a basic idea.

First notice that you have to count the ways for each chunk of **W** in the input and multiply them together to get the total number of ways. For example let's say the input is:

```
GWWGWWB
```

Now the first chunk of **WW** can be replaced with either **RB** or **BR** and the second chunk can be replaced with either **RG**, **BG** or **BR**. So the number of ways will be $2 * 3 = 6$.

Let's focus on one chunk at a time. if we can solve for a single chunk, the other parts are trivial.

Now you probably realized already that the solution for each chunk of **W** depends on three things:

- Number of **W**'s in the chunk
- The letter before the first **W**
- The letter after the last **W**

It doesn't really matter what is the letter before and after the chunk, the only thing that matters is **whether the letters surrounding the chunk is same or not**! That means "GWWWG" is no different from "BWWWG" and "BWWG" is same as "RWWB".

Now we have a integer n (length of the **W** chunk) and a boolean ***isSame,*** we have to find the number of ways to replace the **W**'s with RGB so that no two adjacent letter is the same.

We will define a method ***solve(n, isSame)*** which will give us the answer. Our target is to break it down into small sub-problem and what technique is in this job then recursion?

Let's think what happens if **isSame = true** and we convert the left-most **W into** something else?

- The length of the W-chunk becomes $n - 1$.

- Now the letters surrounding the **W** is not same anymore, that means **isSame = false**. (because you are not allowed to choose the letters which surrounded the chunk).
- The subproblem becomes **solve(n -1, !isSame).**

As you can color the left-most **W** in two ways (one letter is forbidden), the solution for the state ***solve(n, isSame)*** *is:*

```
solve(n, isSame) = 2 * solve(n -1, !isSame) where isSame = true
```

Now what happens if **isSame = false**? We can still choose two different character in place of the leftmost **W,** but now depending on what character you choose, either IsSame will remain false, or it will become true. the solution for the state ***solve(n, !isSame):***

```
solve(n, !isSame) = solve(n -1, !isSame)  + solve(n - 1, isSame) where isSame = false
```

We need to define a base case for the recursion to stop. When there is no more **W** left, the surrounding characters must not be the same. That means if $n = 0$ we will return 1 if isSame = true, otherwise we will return 0. That gives us the following recursion:

```
int solve(int n, boolean isSame) {
    if (n == 0) {
        return !isSame;
    }

    if (isSame) {
        return 2 * solve(n - 1, !isSame);
    } else {
        return solve(n - 1, isSame) + call(n - 1, !isSame);
    }
}
```

The time complexity of the recursive method above is $O(n)$, right? Wrong, the solution is exponential, more then 2^n! This is because every state will be called multiple times. For example the state $(5, 0)$ will call $(4, 1)$, $(4, 0)$ and the state $(5, 1)$ will call $(4, 0)$ and the state $(4, 0)$ will computed twice. As the tree

grows big, each method will be called huge number of times. This property has a fancy name called "Overlapping Subproblem". To avoid this, we just need to memorize each state as we go along, if a state is already computed, just return the answer from the memorization table.

```
int solve(int n, boolean isSame) {
    if (n == 0) {
        return !isSame;
    }

    if (save[n][isSame] != NULL) {
     return save[n][isSame];
    }

    if (isSame) {
        save[n][isSame] = 2 * solve(n - 1, !isSame);
    } else {
        save[n][isSame] = solve(n - 1, isSame) + call(n - 1, !isSame);
    }
    return save[n][isSame];
}
```

I have just saved the answer in a table before returning it and it changes the time complexity to $O(n)$ with 2 constant factor! This technique of breaking down problem and memorizing it is called Dynamic Programming.

Now you know how to solve the main part already, other parts just depends on how you want to implement it. You need to take care of the case where there is nothing on the right or left of the W-chunk which is pretty easy using combinatorics.

That's it for today. Feel free to ask any questions.

SHAFAET ASHRAF

MORE POSTS (HTTP://EN.SHAFAETSPLANET.COM/AUTHOR/SHAFAET/)

PREVIOUS ARTICLE

INTRO TO STAIRCASE NIM (HTTP://EN.SHAFAETSPLANET.COM/STAIRCASE-NIM/)

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

COMMENT

NAME *

EMAIL *

WEBSITE

POST COMMENT