

**My TopCoder****Competitions**[Overview](#)[Copilot Opportunities](#)[Design](#)[Development](#)[UI Development](#)[QA and Maintenance](#)[CloudSpokes](#)[Algorithm](#)[High School](#)[The Digital Run](#)[Submit & Review](#)**TopCoder Networks**[Events](#)[Statistics](#)[Tutorials](#)[Forums](#)[Surveys](#)[My TopCoder](#)[Help Center](#)[About TopCoder](#)

Member Search:

Handle: [Go](#)[Advanced Search](#)[Dashboard](#) > [TopCoder Competitions](#) > ... > [Algorithm](#)
[Problem Set Analysis](#) > [SRM 618](#)[Search](#)

TopCoder Competitions

**SRM 618**[View](#)[Attachments \(18\)](#)[Info](#)[Browse Space](#)Added by [\[\[rng_58\]\]](#), last edited by [vexorian](#) on May 01, 2014 ([view change](#))Labels: (None) [EDIT](#)**Single Round Match 618**

Thursday, April 24th, 2014

[Archive](#)
[Match Overview](#)
[Discuss this match](#)**Match summary****The Problems**[WritingWords](#) | [LongWordsDiv2](#) | [MovingRooksDiv2](#) | [Family](#) | [LongWordsDiv1](#) | [MovingRooksDiv1](#)**LongWordsDiv1**[Rate It](#)[Discuss it](#)

Used as: Division One - Level Two:

Value	500
Submission Rate	139 / 534 (26.03%)
Success Rate	131 / 139 (94.24%)
High Score	shb123 for 499.72 points (0 mins 40 secs)
Average Score	270.64 (for 131 correct submissions)

The size and the patterns

This problem requires us to understand the properties of the strings that have the maximum number of characters for each alphabet size. The problem statement's example, tell us for $n = 1$ and $n = 2$:

- For $n = 1$: A.
- For $n = 2$: ABA, BAB.

Let's try for $n = 3$. You will find many strings of 5 characters, most of them follow a pattern like ABCBA, the only thing that changes is the letters: ABCBA, ACBCA, BACAB, BCACB, CABAC, CBABC. However, there is also another pattern that has 5 characters: ABACA. The strings that do this are: ABACA, ACABA, BABCB, BCBAB, CACBC, CBCAC. One thing useful in showing that these are the largest strings possible is to notice that no valid string can contain a letter more than 3 times (Else the XYXY rule would be broken).

It is necessary to understand why these patterns are the solutions and how to exactly build them.

First of all, it appears that a good strategy to find the maximum-length string is to make a permutation of n letters and then to mirror it:

A B C D ... Z ... D C B A

If this strategy was guaranteed to give the maximum length, then the maximum length is always $2n - 1$. Thus we need to show that this is indeed the maximum length.

We can tell easily that the maximum length is $2n - 1$ for $n = 1, n = 2$ and $n = 3$. Let's try a proof by induction. Assume that we know that for all $n < k$, the maximum length is : $2n - 1$.

When $n = k$, we should start the string with some character, let's name it **a**. So we begin:

a...

We know this letter can appear at most 3 times:

- If we decide to make **a** appear only 1 time, then we only have $n - 1$ alphabet letters available for the remaining positions of the string. We know that the largest string of $n - 1$ alphabet letters is $2(n - 1) - 1$. This makes the total length: $2(n - 1) - 1 + 1 = 2n - 2$.

- If we decide to make 2 **a** characters, then things become more interesting:

aXXX...XaYYY...Y

We are going to show that the YYY...Y section should contain no characters. The first thing to notice is that if a character c belongs to the XXX..X section, it shouldn't be used in the YYY..Y section. If a character appeared in both parts at the same time, then **a-c-a-c** would be a subsequence of the string, and that is forbidden.

Second, XX..X should be non-empty. If it was empty, then we would have two consecutive **a** characters.

Assume YY..Y is empty, then we have $n - 1$ alphabet letters available for the XX..X, the largest length of XX..X is $2(n - 1) - 1$, the total length is: $2n - 1$. This is better than $2n - 2$, so if at all possible, we shouldn't use the first letter only once. (If $n = 1$, it is not possible to use it twice).

If YY..Y is non-empty, then we assign P alphabet letters to XX..X and Q alphabet letters to YY..Y. Note that $P + Q = n - 1$. From inductive logic, the maximum lengths of XX..X and YY..Y are: $2P - 1$ and $2Q - 1$. The total length is: $2P + 2Q = 2(P + Q) = 2(n - 1) = 2n - 2$. But $2n - 1$ is better.

- Finally, we can try to use letter **a** three times:

aXX...XaYY...YaZZ...Z

XX..X and YY..Y cannot be empty. You can use a proof similar to the one above to show that ZZ..Z should be empty. Once again, XX..X and YY..Y cannot share character. We have that there are P alphabet letters used in XX..X and Q alphabet letters used in YY..Y. $P + Q = n - 1$ and the total length is: $2P - 1 + 2Q - 1 + 3 = 2(P + Q) + 2 = 2(n - 1) + 2 = 2n - 1$.

Therefore, we have discovered two things: a) The maximum length is $2n - 1$. b) There are 3 patterns that achieve this maximum length:

- For $n = 1$ only:

a

Where **a** is the only alphabet letter.

- For $n \geq 2$:

aXX...Xa

Where **a** is any alphabet letter, and **XX..X** contains the remaining $n - 1$ alphabet letters.

- For $n \geq 3$:

aXX..XaYY..Ya

Where **XX..X** and **YY..Y** are non-empty partitions of the set of $n - 1$ letters other than **a**.

Dynamic programming

Once we understand the available patterns that maximize the length, we can make a recurrence relation that counts the number of strings that can follow them.

A nice way to simplify the logic is to notice that we can just count the number of ways to shape the pattern and finally multiply by $n!$. It is easy to see why it works for the first kind of patterns. Imagine that for $n = 26$, we find the following pattern:

abcde...z...edcba

There are $26!$ different strings that follow this pattern, example: xwpok..h..kopwx. So we just [permute](#) the alphabet letters.

The real useful realization is that we can do the same for the other patterns, like, for $n = 4$: **abcbada**, there are still $n! = 4! = 4 \cdot 3 \cdot 2 = 24$ ways to permute this pattern.

Finally, the recurrence relation. $f(n)$ counts the number of *patterns* for n alphabet letters. (So the number of *strings* is $f(n) \cdot n!$).

Base case: $f(1) = 1$. There is only one pattern: **a**.

When $n > 1$, we can use the **aXX..Xa** pattern. We should count the number of patterns with $n - 1$ alphabet letters: $f(n - 1)$. So there are $f(n - 1)$ patterns of the kind: **aXX..Xa**

When $n > 2$ we can also use the **aXX..XaYY..Ya** pattern. We need to pick P and Q , the number of alphabet letters assigned to **XX..X** and **YY..Y**, each. Since $P + Q = n - 1$, we can guess that $Q = n - 1 - P$, so there are $O(n)$ options for (P, Q) . We can try them all. For each (P, Q) , we count the number of patterns $f(P)$ and $f(Q)$, since the patterns are independent, the total number of combined patterns is: $f(P) \times f(Q)$. We repeat this for each valid value of P .

There are $O(n)$ possible inputs for $f(n)$ and in each, we might need to run a $f(n)$ loop. Let's use [dynamic programming](#) and the complexity will be $O(n^2)$.

Code

Problems that ask you to return the result modulo 1,000,000,007 (or a similar number) usually do so to allow us to solve problems that have large results without actually using those large numbers in calculations. We can handle these numbers with modular arithmetic. Read [this recipe](#) for more info.

```

const int MOD = 1000000007;

// Calculates x! modulo MOD
long modFactorial(int x) {
    long r = 1;
    for (int i = 2; i <= x; i++) {
        r = (r * i) % MOD;
    }
    return r;
}

long dp[5001];

long count(int n) {
    dp[0] = 1;
    for (int i = 1; i <= n; i++) {
        //1) 2i - 1 = 2 + (2(i-1) - 1)
        dp[i] = dp[i-1];
        //2) find p,q:
        for (int p = 1; p + 2 <= i; p++) {
            dp[i] += (dp[i - p - 1] * dp[p]) % MOD; //multiply f(p)*f(i-1-p)
        }
        dp[i] %= MOD;
    }
    return ( dp[n] * modFactorial(n) ) % MOD;
}

```

Alternative solutions and additional comments.

<Place your comments here>

Next problem: [MovingRooksDiv1](#)

Author



By **vexorian**

TopCoder Member

Editorial feedback	Choose
I liked it.	<input checked="" type="radio"/>
I didn't like it.	<input checked="" type="radio"/>

Comments ([Hide Comments](#))


In Java,
 You do not need to convert between the character and its numeric representation.
 Java will use the Unicode representation in case of arithmetic operations.
 So, the following line will work also in Java:
 sum += ch - 'A' + 1;



Posted by alhussain at [May 04, 2014 17:57](#) | [Reply To This](#)

In the C++ solution of the "Family" problem,
`int color [50];` should be
`int color [100];`
 as it mentioned in the problem statement,
 parent1 will contain between 1 and 100 elements, inclusive. and
 For each i, the i-th element (0-based) of parent1 will be between -1 and i-1
 inclusive

Please correct me, If I misunderstood any point. Thanks!

 Posted by tausiq at [May 10, 2014 04:27](#) | [Reply To This](#)

Yes.

 Posted by vexorian at [May 10, 2014 08:58](#) | [Reply To This](#)

 [Add Comment](#)

[Home](#) | [About TopCoder](#) | [Press Room](#) | [Contact Us](#) | [Privacy](#) | [Terms](#)
[Developer Center](#) | [Corporate Services](#)

Copyright TopCoder, Inc. 2001-2014