Kick Start 2022 - Round C

Analysis: Palindromic Deletions

The expected value of a discrete random variable is defined as the weighted average over all possible values of the variable. That is, for a discrete random variable X with probability function $P(X)$, its expected value $E[X] =$ the sum of $X \times P(X)$ over all possible values of X.

Keeping the above definition in mind, we can define a random variable X as the number of palindromes encountered in a particular game of Palindromic Deletions on a string of length $\mathbf{N}$, with $P(X)$ being the probability function. We can write $P(X) = \frac{A(X)}{B}$, where $A(X)$ is the number of distinct games which generate X palindromes and B is the total number of distinct games. Notice that the number of distinct games can be defined as the total number of orders of picking indexes 1 to $\mathbf{N}$, which is equivalent to to the number of permutations of an array of size $\mathbf{N}$. Therefore, $B = \mathbf{N}!$.

With the above simplification, we can write the expected value $E[X] =$ the sum of $\frac{X \times A(X)}{\mathbf{N}!}$ over all possible integer values of X between 1 and $\mathbf{N}$. Note that the game counts an empty string as a palindrome, hence it is not possible for X to be 0. The problem translates into calculating the sum of $X \times A(X)$ over all X under modulo $10^9 + 7$ (let us call this value Y). We can then multiply Y by the modular inverse of $\mathbf{N}!$ under modulo $10^9 + 7$ to get $E[X]$. Note that we are able to simply multiply by the modular inverse of $\mathbf{N}!$ modulo $10^9 + 7$ without reducing the fraction to an irreducible form since $10^9 + 7$ is prime and $\mathbf{N} \leq 400$, which gives us $\gcd(\mathbf{N}!, 10^9 + 7) = 1$ (as the denominator will not contain any prime factors > 400).

Test Set 1

For $\mathbf{N} \leq 8$, we can simply recursively generate all possible games to calculate Y. One way to do this is by starting with the input string and a count of palindromes encountered $cnt = 0$. In each recursion level, we delete a character from the string and check if the new string is a palindrome. If the new string is a palindrome, we increase cnt by 1. We recursively do this operation again with the new string and cnt. This is done for each character in the string at a particular recursion level. We return when we reach an empty string, at which point we add cnt to an outer sum variable. After the top level recursive function returns, sum will store the value Y. All operations are done under modulo $10^9 + 7$.

For example, let us visualize this with the sample string aba.

1. "<u>a</u>ba", $0 \rightarrow$ "<u>b</u>a", $0 \rightarrow$ "<u>a</u>", $1 \rightarrow$ "", 2
2. "<u>a</u>ba", $0 \rightarrow$ "b<u>a</u>", $0 \rightarrow$ "<u>b</u>", $1 \rightarrow$ "", 2
3. "a<u>b</u>a", $0 \rightarrow$ "<u>a</u>a", $1 \rightarrow$ "<u>a</u>", $2 \rightarrow$ "", 3
4. "a<u>b</u>a", $0 \rightarrow$ "a<u>a</u>", $1 \rightarrow$ "<u>a</u>", $2 \rightarrow$ "", 3
5. "ab<u>a</u>", $0 \rightarrow$ "<u>a</u>b", $0 \rightarrow$ "<u>b</u>", $1 \rightarrow$ "", 2
6. "ab<u>a</u>", $0 \rightarrow$ "a<u>b</u>", $0 \rightarrow$ "<u>a</u>", $1 \rightarrow$ "", 2

Adding all cnt at the end of each simulation, we get $sum = 2 + 2 + 3 + 3 + 2 + 2 = 14$. Hence, we get an expected value of $\frac{14}{3!} = \frac{14}{6}$. Dividing 14 and 6 by their GCD (greatest common divisor) i.e. 2, we get the irreducible fraction $\frac{7}{3}$.

The number of orders generated is $\mathbf{N}!$, while it takes $O(\mathbf{N})$ to check if a string is a palindrome. The time and space complexity comes out to be $O(\mathbf{N} \times \mathbf{N}!)$, which is approximately 3×10^5

operations.

Test Set 2

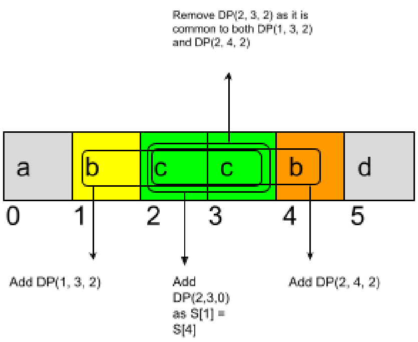
Instead of thinking about Y as the sum of $X \times A(X)$, where $A(X)$ is the number of games that generate X palindromes, we can conversely think about it as the sum of $Q(K)$ over all $K = 0$ to $\mathbf{N} - 1$, where $Q(K)$ is the number of games in which a palindrome of length K is encountered.

Another observation to make here is that any palindrome that we encounter in a game will be a subsequence of the input string $\mathbf{S}$. Consider a palindrome of length K that exists as a subsequence of the input string $\mathbf{S}$. The number of games in which we will encounter this palindrome is $K! \times (\mathbf{N} - K)!$. This is because to encounter a particular palindrome of length K in a game, we must first remove the $\mathbf{N} - K$ characters that are not part of the palindrome in any order, then remove the remaining K characters in any order. With this, we get $Q(K) = K! \times (\mathbf{N} - K)! \times F(K)$, where $F(K)$ is the number of palindromes of length K that occur as a subsequence of $\mathbf{S}$.

All that remains is to find the number of palindromes of length K that occur as a subsequence of $\mathbf{S}$. We can use dynamic programming to do this. We define our state as (L, R, len), where the value of this state $DP(L, R, len)$ equals the number of palindromes of length len that can be found as a subsequence of the substring $\mathbf{S}[L, R]$ (indices L and R inclusive). We have three base cases:

- Any state with $len = 0$ would have a value of 1.
- Any state with $len < 0$ would have a value of 0.
- Any state with $L > R$ that does not satisfy any of the above two cases would have a value of 0.

Now to calculate $DP(L, R, len)$, notice that if $\mathbf{S}[L] = \mathbf{S}[R]$, then we have $DP(L + 1, R - 1, len - 2)$ palindromes that have $\mathbf{S}[L]$ and $\mathbf{S}[R]$ as the first and last character respectively (or $DP(L + 1, R - 1, len - 1)$ palindromes if $L = R$). All remaining palindromes can be found as a union of the palindromes of length len found in substrings $\mathbf{S}[L, R - 1]$ and $\mathbf{S}[L + 1, R]$. By the inclusion-exclusion principle, the value of this union comes out to be $DP(L, R - 1, len) + DP(L + 1, R, len) - DP(L + 1, R - 1, len)$. We subtract the palindromes found in $DP(L + 1, R - 1, len)$ as we would be double counting them in $DP(L, R - 1, len)$ and $DP(L + 1, R, len)$. The figure below helps visualize $DP(1, 4, 2)$ for string $abccbd$. Strings are zero indexed.



With the above defined state and DP calculation in mind, we get $F(K) = DP(0, \mathbf{N} - 1, K)$ and $Q(K) = DP(0, \mathbf{N} - 1, K) \times K! \times (\mathbf{N} - K)!$. Finally, we get Y by summing all $Q(K)$ for $K = 0$ to $\mathbf{N} - 1$.

We can precompute factorials upto 400 linearly, then subsequently retrieve the precomputed factorial values in constant time. The time to compute all states of the DP is $O(\mathbf{N}^3)$. Similarly, we have $\mathbf{N}^3$ integer DP states, giving us an overall time and space complexity of $O(\mathbf{N}^3)$.

Bonus Solution

While the above approach is good enough to pass the constraints if the DP array uses 32-bit integers, we can further optimize on space by optimizing the calculation for $F(K)$ for each K. Instead of creating an $\mathbf{N} \times \mathbf{N} \times \mathbf{N}$ DP array, we can create three $\mathbf{N} \times \mathbf{N}$ DP arrays, where the first corresponds to some palindrome length len, the second corresponds to length $len - 1$ and the third corresponds to length $len - 2$. As the transitions described in the previous approach for the DP values of a given length len just need DP values from $len - 1$ and $len - 2$, we can construct the DP array for len using DP values of $len - 1$ and $len - 2$, then construct the DP array for $len + 1$ using values from len and $len - 1$, then for $len + 2$ using values from $len + 1$ and len, and so on. In this way, we can start with DP arrays for $len = 0$ and 1 as base cases and iteratively construct for each $len = 2$ to $\mathbf{N} - 1$. The rest of the idea is similar to the previous solution. With this approach, even though the time complexity is still $O(\mathbf{N}^3)$, the space complexity comes down to $O(\mathbf{N}^2)$.