HARD 2:

You are given a string s. You can convert s to a

palindrome by adding characters in front of it.

Return the shortest palindrome you can find by performing this transformation.

SOLUTION:

public class ShortestPalindrome {

public static String shortestPalindrome(String s) {

int n = s.length();

if (n <= 1) {

return s;

}

String modifiedString = s + "#" + new StringBuilder(s).reverse().toString();

int[] kmpTable = computeKMPTable(modifiedString);

int palindromeLength = kmpTable[modifiedString.length() - 1];

String suffix = s.substring(palindromeLength);

String prefix = new StringBuilder(suffix).reverse().toString();

return prefix + s;

}

private static int[] computeKMPTable(String pattern) {

int[] kmpTable = new int[pattern.length()];

int j = 0;

for (int i = 1; i < pattern.length(); i++) {

if (pattern.charAt(i) == pattern.charAt(j)) {

j++;

kmpTable[i] = j;

} else {

if (j != 0) {

j = kmpTable[j - 1];

i--;

} else {

kmpTable[i] = 0;

}

}

}

return kmpTable;

}

public static void main(String[] args) {

System.out.println(shortestPalindrome("aacecaaa")); // Output: "aaacecaaa"

System.out.println(shortestPalindrome("abcd")); // Output: "dcbabcd"

}

}

SUMMARY AND ALGORITHM:

Summary:

The provided Java code defines a class named `ShortestPalindrome` with a method `shortestPalindrome` that takes a string `s` and returns the shortest palindrome that can be formed by adding characters to the beginning of `s`. The solution involves using the Knuth-Morris-Pratt (KMP) algorithm to efficiently find the length of the longest palindromic prefix in a modified string.

Algorithm:

1. `shortestPalindrome` Method:

- Check if the length of the input string `s` is less than or equal to 1. If true, return the input string as it is already a palindrome.

- Create a modified string by concatenating the input string with a special character '#' and its reverse.

- Compute the KMP table for the modified string using the `computeKMPTable` method.

- Retrieve the length of the longest palindromic prefix from the last element of the KMP table.

- Extract the suffix of the input string starting from the palindromic prefix length.

- Reverse the suffix to obtain the prefix.

- Concatenate the prefix with the original string and return the result.

2. `computeKMPTable` Method:

- Initialize an array `kmpTable` of the same length as the input pattern string.

- Iterate over the pattern string from the second character.

- If the characters at the current positions match, increment `j` and set `kmpTable[i]` to `j`.

- If the characters do not match and `j` is not zero, update `j` to the value at the previous index in the KMP table (`kmpTable[j - 1]`), and decrement `i` to recheck the current character in the next iteration.

- If the characters do not match and `j` is already zero, set `kmpTable[i]` to 0.

- Return the computed KMP table.

Example Test Cases and Expected Outputs:

System.out.println(shortestPalindrome("aacecaaa")); // Output: "aaacecaaa"

System.out.println(shortestPalindrome("abcd")); // Output: "dcbabcd"

These test cases demonstrate the ability of the `shortestPalindrome` method to find the shortest palindrome by efficiently utilizing the KMP algorithm to compute the length of the longest palindromic prefix in the modified string.