Day 60/180 Strings HARD Problems

1:Minimum characters to be added at front to make string palindrome:
The solution and code both are explained in the video.

```
// Function to find the Z-function of a string
vector<int> zfind(string s) {
    int m = s.length();
    vector<int> res(m, 0); // Initialize the result vector with zeros
    int j = 1, i = 0;

    // Iterate through the string
    while (j < m) {
        if (s[i] == s[j]) {
            res[j] = i + 1; // If characters match, update the result and move both
pointers
            i++, j++;
        } else {
            if (i > 0) {
                i = res[i - 1]; // If characters do not match and there is a previous
match, jump to that position
            } else {
                j++; // If characters do not match and no previous match, move to the
next character
            }
        }
    }

    return res; // Return the Z-function of the string
}

// Function to compute the Longest Palindromic Suffix (LPS) array using the
Z-function
vector<int> lps(string s) {
    vector<int> z = zfind(s); // Compute the Z-function of the string
    string s1 = s;
    reverse(s1.begin(), s1.end()); // Reverse the string to find the palindromic
suffix
    int n = s.length();
    vector<int> lps(n, 0); // Initialize the LPS array with zeros
    int j = 0, i = 0;

    // Iterate through the reversed string
    while (j < n) {
        if (s[i] == s1[j]) {
            lps[j] = i + 1; // If characters match, update the LPS array and move
both pointers
            i++, j++;
        } else {
            if (i > 0) {
                i = z[i - 1]; // If characters do not match and there is a previous
match, jump to that position
            } else {
                j++; // If characters do not match and no previous match, move to the
next character
            }
        }
    }

    reverse(lps.begin(), lps.end()); // Reverse the LPS array to match the original
string
    return lps; // Return the Longest Palindromic Suffix array
}

// Function to find the minimum characters to be added to make the string a
palindrome
int minChar(string s) {
    int n = s.length();
    vector<int> z = lps(s); // Compute the Longest Palindromic Suffix array
    int cur = 0;

    // Iterate through the LPS array to find the maximum overlap
    for (int i = 0; i < n; i++) {
        if (z[i] >= i) {
            cur = i + z[i]; // Update the current position based on the LPS array
        }
    }

    return n - cur; // Return the minimum characters to be added to make the string a
palindrome
}
```

Explanation

The zfind function computes the Z-function of a given string, which represents the length of the longest common prefix of a string and its suffix starting at each position.

The lps function uses the Z-function to compute the Longest Palindromic Suffix (LPS) array, which represents the length of the longest suffix that is also a palindrome for each position in the string.

The minChar function uses the LPS array to find the minimum number of characters to be added to the given string to make it a palindrome. It iterates through the LPS array to find the maximum overlap, and the result is obtained by subtracting this overlap from the total length of the string.

2:Repeated String Match:

The solution and code both are explained in the video.

3: Minimum Appends for Palindrome!

This question is similar to first question, in first question we have add first here we have to append in last so we can reverse the string and apply same algo as first question.

```cpp
// Function to compute the Z-function of a string
vector<int> zfind(string s) {
    int m = s.length();
    vector<int> res(m, 0); // Initialize the result vector with zeros
    int j = 1, i = 0;

    // Iterate through the string
    while (j < m) {
        if (s[i] == s[j]) {
            res[j] = i + 1; // If characters match, update the result and move both
pointers
            i++, j++;
        } else {
            if (i > 0) {
                i = res[i - 1]; // If characters do not match and there is a previous
match, jump to that position
            } else {
                j++; // If characters do not match and no previous match, move to the
next character
            }
        }
    }

    return res; // Return the Z-function of the string
}

// Function to compute the Longest Palindromic Suffix (LPS) array using the
Z-function
vector<int> lps(string s) {
    vector<int> z = zfind(s); // Compute the Z-function of the string
    string s1 = s;
    reverse(s1.begin(), s1.end()); // Reverse the string to find the palindromic
suffix
    int n = s.length();
    vector<int> lps(n, 0); // Initialize the LPS array with zeros
    int j = 0, i = 0;

    // Iterate through the reversed string
    while (j < n) {
        if (s[i] == s1[j]) {
            lps[j] = i + 1; // If characters match, update the LPS array and move
both pointers
            i++, j++;
        } else {
            if (i > 0) {
                i = z[i - 1]; // If characters do not match and there is a previous
match, jump to that position
            } else {
                j++; // If characters do not match and no previous match, move to the
next character
            }
        }
    }

    reverse(lps.begin(), lps.end()); // Reverse the LPS array to match the original
string
    return lps; // Return the Longest Palindromic Suffix array
}

// Function to find the minimum characters to be added to make the string a
palindrome
int minChar(string s) {
    int n = s.length();
    vector<int> z = lps(s); // Compute the Longest Palindromic Suffix array
    int cur = 0;

    // Iterate through the LPS array to find the maximum overlap
    for (int i = 0; i < n; i++) {
        if (z[i] >= i) {
            cur = i + z[i]; // Update the current position based on the LPS array
        }
    }

    return n - cur; // Return the minimum characters to be added to make the string a
palindrome
}

// Function to solve the problem
```

```
int Solution::solve(string s) {
    reverse(s.begin(), s.end()); // Reverse the input string
    return minChar(s); // Call the minChar function to find the solution
}
```

4:SHortest Pallindrome

```
class Solution {
public:
    // Function to compute the Z-function of a string
    vector<int> findZ(string s) {
        vector<int> a(s.length(), 0);
        int i = 0, j = 1;

        // Iterate through the string to compute the Z-function
        while (j < s.length()) {
            if (s[i] == s[j]) {
                a[j] = i + 1; // If characters match, update the Z-function and
move both pointers
                i++, j++;
            } else {
                if (i > 0)
                    i = a[i - 1]; // If characters do not match and there is a
previous match, jump to that position
                else
                    j++;
            }
        }

        return a; // Return the Z-function of the string
    }

    // Function to get the Z-function of one string with respect to another
    vector<int> get(string &a, string &b) {
        int i = 0, j = 0;
        vector<int> ans(a.length(), 0);
        vector<int> z = findZ(a); // Compute the Z-function of the first string

        // Iterate through the second string to find the Z-function with respect to
the first string
        while (j < a.length()) {
            if (a[i] == b[j]) {
                ans[j] = i + 1; // If characters match, update the result and move
both pointers
                i++, j++;
            } else {
                if (i > 0)
                    i = z[i - 1]; // If characters do not match and there is a
previous match, jump to that position
                else
                    j++;
            }
        }

        reverse(ans.begin(), ans.end()); // Reverse the result to match the
original string
        return ans; // Return the Z-function with respect to the first string
    }

    // Function to find the shortest palindrome by using Z-function
    string shortestPalindrome(string s) {
        int n = s.length();
        string b = s;
        reverse(b.begin(), b.end());
        vector<int> cur = get(s, b); // Get the Z-function with respect to the
reversed string
        int j;

        // Iterate to find the maximum overlap
        for (int i = n - 1; i >= 0; i--) {
            if (cur[i] >= i + 1) {
                if (cur[i] == i + 1)
```

```
                    j = 2 * i;
                else
                    j = 2 * i + 1;
                break;
            }
        }

        string extra = "";

        if (j < s.length())
            extra = s.substr(j + 1); // Extract the extra characters needed to form
the palindrome
        reverse(extra.begin(), extra.end());
        string res = extra + s; // Concatenate the extra characters and the
original string
        return res; // Return the shortest palindrome
    }
};
```