Count Different Palindromic Subsequences

Given a string S, find the number of different non-empty palindromic subsequences in S, and **return that number modulo `10^9 + 7`.**

A subsequence of a string S is obtained by deleting 0 or more characters from S.

A sequence is palindromic if it is equal to the sequence reversed.

Two sequences `A_1, A_2, ...` and `B_1, B_2, ...` are different if there is some `i` for which `A_i != B_i`.

Example 1:

```
Input: 
S = 'bccb'
Output: 6
Explanation: 
The 6 different non-empty palindromic subsequences are 'b', 'c', 'bb', 'cc', 'bcb', '
bccb'.
Note that 'bcb' is counted only once, even though it occurs twice.
```

Example 2:

```
Input: 
S = 'abcdabcdabcdabcdabcdabcdabcdabcddcbadcbadcbadcbadcbadcbadcbadcba'
Output: 104860361
Explanation: 
There are 3104860382 different non-empty palindromic subsequences, which is 104860361
 modulo 10^9 + 7.
```

Note:

- The length of `S` will be in the range `[1, 1000]`.
- Each character `S[i]` will be in the set `{'a', 'b', 'c', 'd'}`.

Solution 1

```java
class Solution {
    int div=1000000007;
    public int countPalindromicSubsequences(String S) {
        TreeSet[] characters = new TreeSet[26];
        int len = S.length();

        for (int i = 0; i < 26; i++) characters[i] = new TreeSet<Integer>();

        for (int i = 0; i < len; ++i) {
            int c = S.charAt(i) - 'a';
            characters[c].add(i);
        }
        Integer[][] dp = new Integer[len+1][len+1];
         return memo(S,characters,dp, 0, len);
    }

    public int memo(String S,TreeSet<Integer>[] characters,Integer[][] dp,int start,
int end){
        if (start >= end) return 0;
        if(dp[start][end]!=null) return dp[start][end];

            long ans = 0;

            for(int i = 0; i < 26; i++) {
                Integer new_start = characters[i].ceiling(start);
                Integer new_end = characters[i].lower(end);
              if (new_start == null || new_start >= end) continue;
                 ans++;
                if (new_start != new_end) ans++;
                ans+= memo(S,characters,dp,new_start+1,new_end);

            }
            dp[start][end] = (int)(ans%div);
            return dp[start][end];
    }

}
```

written by kay_deep original link here

Solution 2

Let `dp(i, j)` be the answer for the string `T = S[i:j+1]` including the empty sequence. The answer is the number of unique characters in `T`, plus `dp(next('a', i) + 1, prev('a', j) - 1)` representing palindromes of the form `"a_a"` where _ is zero or more characters, plus `dp(next('b', i) + 1, prev('b', j) - 1)` representing `"b_b"`, etc.

Here, `next('a', i)` means the next index at or after `i` where `A[next('a', i)] = 'a'`, and so on.

```
class Solution(object):
    def countPalindromicSubsequences(self, S):
        N = len(S)
        A = [ord(c) - ord('a') for c in S]
        prv = [None] * N
        nxt = [None] * N

        last = [None] * 4
        for i in xrange(N):
            last[A[i]] = i
            prv[i] = tuple(last)

        last = [None] * 4
        for i in xrange(N-1, -1, -1):
            last[A[i]] = i
            nxt[i] = tuple(last)

        MOD = 10**9 + 7
        memo = [[None] * N for _ in xrange(N)]
        def dp(i, j):
            if memo[i][j] is not None:
                return memo[i][j]
            ans = 1
            if i <= j:
                for x in xrange(4):
                    i0 = nxt[i][x]
                    j0 = prv[j][x]
                    if i <= i0 <= j:
                        ans += 1
                    if None < i0 < j0:
                        ans += dp(i0+1, j0-1)
            ans %= MOD
            memo[i][j] = ans
            return ans

        return dp(0, N-1) - 1
```

written by awice original link here

Solution 3

Let dp[len][i][x] be the number of distinct palindromes of the subtring starting at i of length len, where the first (and last) character is x. The DP formula is simple :

- If s[i] != x, then dp[len][i][x] = dp[len-1][i+1][x] (ignoring the first character in this window)
- If s[i+len-1] != x then dp[len][i][x] = dp[len-1][i][x] (ignoring the last character in this window)
- If both the first and last characters are x, then we need to count the number of distinct palindromes in the sub-window from i+1 to i + len -2. Need to be careful with how we count empty string.
  Since we only need to subproblems of length len-1, len-2, we only need to remember the solutions for the subproblems of length len, len-1, len-2. This is needed to pass the max test case.

```
class Solution {
public:
    int countPalindromicSubsequences(string s) {
        int md = 1000000007;
        int n = s.size();
        int dp[3][n][4];
        for (int len = 1; len <=n; ++len) {
            for (int i = 0; i + len <=n; ++i) for (int x = 0; x < 4; ++x)  {
                int &ans = dp[2][i][x];
                ans = 0;
                int j = i + len - 1;
                char c = 'a' + x;
                if (len == 1) ans = s[i] == c;
                else {
                    if (s[i] != c) ans = dp[1][i+1][x];
                    else if (s[j] != c) ans = dp[1][i][x];
                    else {
                        ans = 2;
                        if (len > 2) for (int y = 0; y < 4;++y) {
                            ans += dp[0][i+1][y];
                            ans %=md;
                        }
                    }
                }
            }
            for (int i=0;i<2;++i) for (int j = 0; j < n; ++j) for (int x=0; x < 4;+
+x)
                dp[i][j][x] = dp[i+1][j][x];
        }
        int ret = 0;
        for (int x = 0; x < 4;++x) ret = (ret + dp[2][0][x]) %md;
        return ret;
    }
};
```

written by elastico original link here