DAA Assignment 4

GROUP 17
PRAVALLIKA KODI (IIT2019234)
VISHWAM SHRIRAM MUNDADA (IIT2019235)
NOONSAVATH SRAVANA SAMYUKTA (IIT2019236)
B.tech Information Technology business Informatics
Indian Institute of Information Technology, Allahabad

28 March 2021

0.1 Question

Given a string S, count the number of non-empty sub strings that are palindromes. A sub string is any continuous sequence of characters in the string. A string is said to be palindrome, if the reverse of the string is same as itself. Two sub strings are different if they occur at different positions in S. Solve using Dynamic programming.

0.2 Abstract

The project aims at counting number of non-empty sub strings that are palindromes in a given string.

0.3 Introduction

Here, different approaches are analysed and used to achieve results. These approaches are:
1) Brute force
2) Dynamic programming

In this report we will explain our solution approach. We explain our code in detail. We will discuss the time complexity analysis and the space complexity analysis. And last but not least, the conclusion.

0.4 Code Explanation

0.4.1 Brute force

This solution involves two nested loops the first loop goes from 0 to size of the string and second loop iterates from i to size of the string and thus two loops cover all the possible substrings starting from each index then for each substring we will check if it is palindrome or not we will be increasing the count variable if we the substring is a palindrome count will be our final answer

Algorithm 1 Brute force

```
function MAIN()
   string s
   Input s

   print COUNTPALINDROMES(s)

function COUNTPALINDROMES(s)
   count <- 0

   for i <- 0 to length[s]
      for j <- i to length[s]
         if ISPALINDROME(s, i, j)
            count <- count+1

   return count

function ISPALINDROME(s, st, end)
   mid <- (end-st+1)/2

   for i <-0 to mid
      if(s[st+i] != s[end-i])
         return false

   return true
```

0.4.2 Dynamic programming

In this DP approach we will maintain a boolean 2D array dp the dp[i][j] represents a substring from index i to j of the string and $j >= i$ condition will be maintained. if i = j then the length of substring is 1 and it is always a palindrome so dp[i][i] will be true if j = i+1 then the length of substring will be 2 and we will just check if s[i] = s[j] and mark it true accordingly in other cases we will just check if the extremes of the substrings are equal or not if they are equal then we will check for middle portion by looking at dp[i+1][j-1] we will be counting trues and get our answer

Algorithm 2 DP

```
function MAIN()
   string s
   Input s

   print COUNTPALINDROMES(s)

function COUNTPALINDROMES(s)
   n <- length[s]
   count <- 0
   bool dp[n][n]
   MEMSET(dp, false, SIZEOF(dp))

   for gap <- 0 to n
      for i <- 0 to n-gap
         j <- i+gap

         if gap = 0
            dp[i][j] = true

         else if gap = 1
            dp[i][j] = s[i] == s[j]

         else
            dp[i][j] = s[i] == s[j]
               && dp[i+1][j-1]

         if(dp[i][j])
            count <- count+1

   return count
```

0.5 Algorithm Analysis

0.5.1 Brute force

In case of brute force algorithm we use nested loop to traverse through the string for all possible substrings. Now inside this nested loop we check if that substring is palindrome. To check whether if it is palindromic substring, we use a for loop.
So, totally we use nested loop which consists of 3 'for' loops, therefore the time complexity is $O(N^3)$.

Time Complexity: $O(N^3)$
$SpaceComplexity : O(1)$

O(1). We don't need to allocate any extra space since we are repeatedly iterating on the input string itself.

0.5.2 Dynamic programming

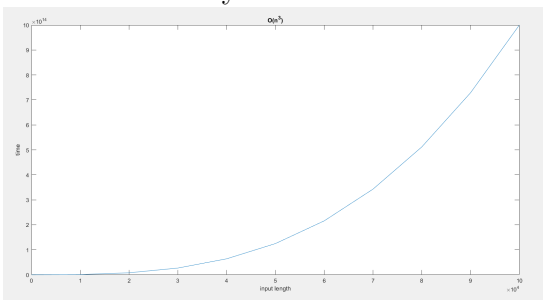
In case of dynamic programming approach, we create a dp table (dp[n][n]), we traverse through the table and give the values as discussed in previous slides. So, for traversing we use a nested loop.So the time complexity O(N^2)
$TimeComplexity : O(N^2)$

As we have created a table of size n x n, space complexity is also O(N^2)

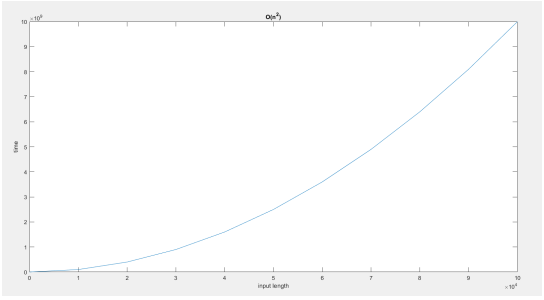
Space Complexity: O(N^2)

0.6 Graph analysis

Brute force
In the brute force as n increases, the time increases by n^3.



Dynamic programming
In the Dynamic programming as n increases, the time increases by n^2.



0.7 Conclusion

From the experimental study we concluded that the average running time of Dynamic programming algorithm is best, which can be observed from the mutual graph of Dynamic programming algorithm and Brute Force algorithm as shown.

0.8 References

https://www.geeksforgeeks.org/longest-palindrome-substring-set-1/
/
https://www.google.com/amp/s/www.geeksforgeeks.org/count-palindrome-sub-str amp/

0.9 code

https://ideone.com/5j8HQW https://ideone.com/ZkKHwv