Design and Analysis of Algorithms

Group-2

Nidhi kamewar IIT2019189

Rishi Mukesh Gupta IIT2019190

Pechetti Venkata Karthik IIT2019191

Abstract:-Given a string S we have to count the number of non-empty palindromic substrings using Dynamic programming

I. INTRODUCTION

In this problem,we have to find the count of all the substrings of a given string which are palindromic.
As the given problem has Overlapping subproblems of recalculating the palindromic substrings of substrings in the given string and Optimal substructure property i.e the given problem can be solved by solving the subproblems of the problem
Hence We can solve the given problem using Dynamic programming.
This report further contains -
II. Algorithm Design
III. Algorithm Analysis
IV. Experimental Study
V. Conclusion

II.ALGORITHM DESIGN

The algorithm we designed is based on dynamic programming **Memorization(Top -Down)** because at first we check if the answer to the subproblem is already computed or not in the lookup table if present we return the value or else we store the result of the problem(checking whether a substring of the string is true or false) in the (lookup table) 2D array.

1.First,string S is passed as an input to the function **countSubstrings**

2.The function initialises all the values of the lookup table to -1 which represents that result of the subproblem is not yet computed

3.We initialise the value of count to 0 which indicates the number of palindromes of the string which will be incremented when a palindromic substring is found.

4.The function iteratively checks the substrings by considering the strings from i to j where i will range from 0 to n-1 and j will range from i to n-1 respectively by calling the function isPal

5.which checks if the substring of S between i to j is a substring or not by checking the lookup table and
returning the respective value or computing the result and storing it in the lookup table.

6. If the starting and the ending character of the string is the same then it will recursively call itself to check whether the substring from i+1 to j-1 is palindrome or not i.e, the substring in the middle.

7.If The value of the starting index is greater then the ending index then it represents that the starting and ending characters of the substrings are matching at each recursive call of the funcion

8.The respective result will be stored in the lookup table.

9.If the substring is a palindrome then the value of count is incremented by 1 it repeats till all the substring are checked and the value of the count is returned

For example :

Consider the string aabcca as S when passed to the function

On each iteration the substrings

i=0;

“a”,“aa”,“aab”,“aabc”,“aabcc”, “aabcca” ;

i=1;

“a”,“ab”,“abc”,abcc”,“abcca”;

i=2

“b”,“bc”,“bcc”,“bcca”;

i=3

“c”,“cc”,”cca”;

i=4

“c”,“ca”;

i=5

“a”

are checked if they are palindromic

Lookup table:

	0	1	2	3	4	5
0	1	1	0	0	0	0
1	-1	1	0	0	0	0
2	-1	-1	1	0	0	0
3	-1	-1	-1	1	1	0
4	-1	-1	-1	-1	1	0
5	-1	-1	-1	-1	-1	1

Here in the above example the strings like **abcc** are not recomputed as the result of the subproblem is already computed when calculating the result of a**abcc**a.

When the algorithm checks strings aa,cc the value of count is incremented by 1 and finally reaches 8 which is the total number of palindromic substrings of the given string “aabcca”.**i.e,a,a,b,c,c,c,a,aa,cc**

PSEUDO CODE

Algorithm

```
int dp[1001][1001];
Function isPal(string s, int i, int j):

        if i > j then
          return 1;
        end if
        if dp[i][j] != -1 then
          return dp[i][j];
        end if
        if s[i] != s[j]  then
          return dp[i][j] = 0;
        end if
        return dp[i][j] = isPal(s, i + 1, j - 1);

 Function countSubstrings(string s):

        memset(dp, -1, sizeof(dp));
        int n ← s.length();
        int count ← 0;
        for  i ←  0 to n-1 do
           for j ←  i  to n-1 do
                  if isPal(s, i, j) then
                    count++;
                  end if
        return count;

  Function main():
          Get s;
          Print(countSubstrings(s))
```

III. ALGORITHM ANALYSIS

Time complexity

The algorithm will have a **O(n^2)** time complexity where n is the length of the string. This is because all possible substrings are being visited and are checked whether they are a palindrome or not (in all the best,avg,worst cases). The total number of substrings is n*(n+1)/2 which is in the order of n^2. Using the memorization table, it can be checked whether a substring is palindrome or not in O(1). Hence, the total time complexity is **O(n^2)**.

Space complexity

The algorithm will have a **O(n^2)** space complexity where n is the length of the string. This is because a memorization table (nxn 2d array) is used to store the results of the subproblems.

Therefore, **O(n^2) auxiliary space** is required.

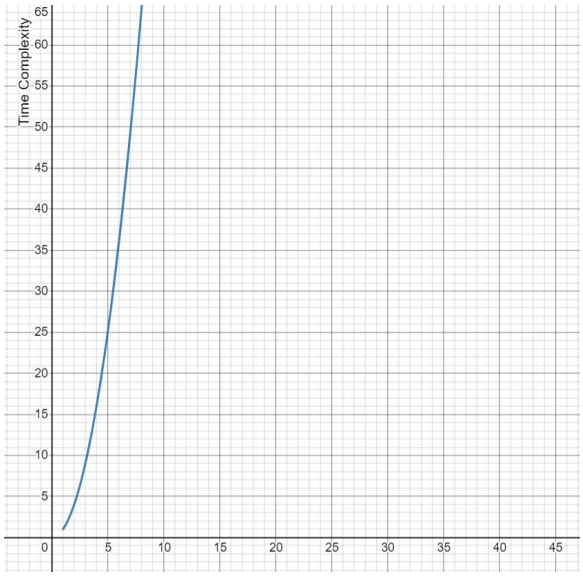

Fig. 1 .Time complexity curve

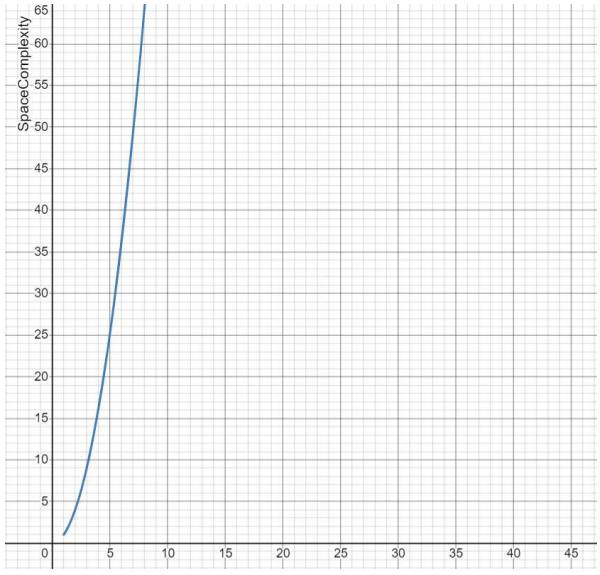

Fig. 2 .Space complexity curve

The time complexity curve when N (string length against Time taken) should be similar to the graph(Fig.1)(parabola) between the Size of the given string(N) and time(y-axis)
Which represents the time complexity of the Algorithm to be **O(n^2)**

The space complexity curve when N(string length against Space(lookup table size)) should be similar to the graph(Fig.2)(parabola)
Which represents the **Space complexity** of the Algorithm to be **O(n^2)** which is the auxiliary space required for the lookup table

IV.EXPERIMENTAL STUDY

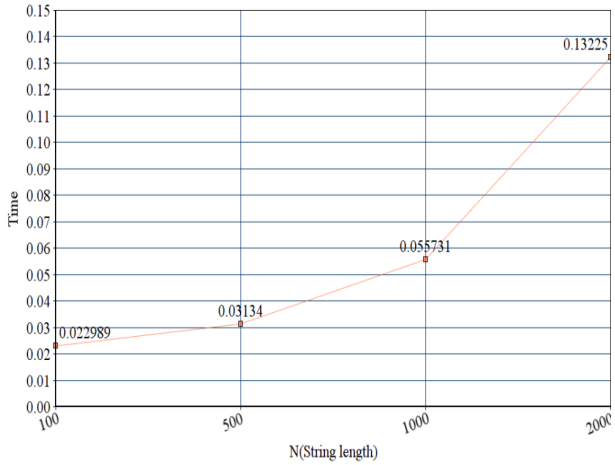

Fig.3 Plotted Time complexity curve(N vs Time)

N(string length)	Time(in seconds)
100	0.022989
500	0.031340
1000	0.055731
2000	0.132250

From the plotted graph of test cases (string length:x-axis) against time taken(y-axis) we can see the time complexity of the graph to be O(n^2)(by asymptotic analysis)

V.CONCLUSION

In this paper we proposed an solution to calculate the number of palindromic substring of a given string using dynamic programming approach.The approach has a time complexity of O(n^2) which is found through study and asymptotic analysis of the graph

REFERENCES

[1] https://www.geeksforgeeks.org
[2] Introduction to Algorithms (CLRS)