Manacher's Algorithm

Longest Palindrome Algorithm

—

By Payal

But the Question is what is a palindrome ?

A palindrome is a sequence of letter that is same when written in reverse order.

Example :

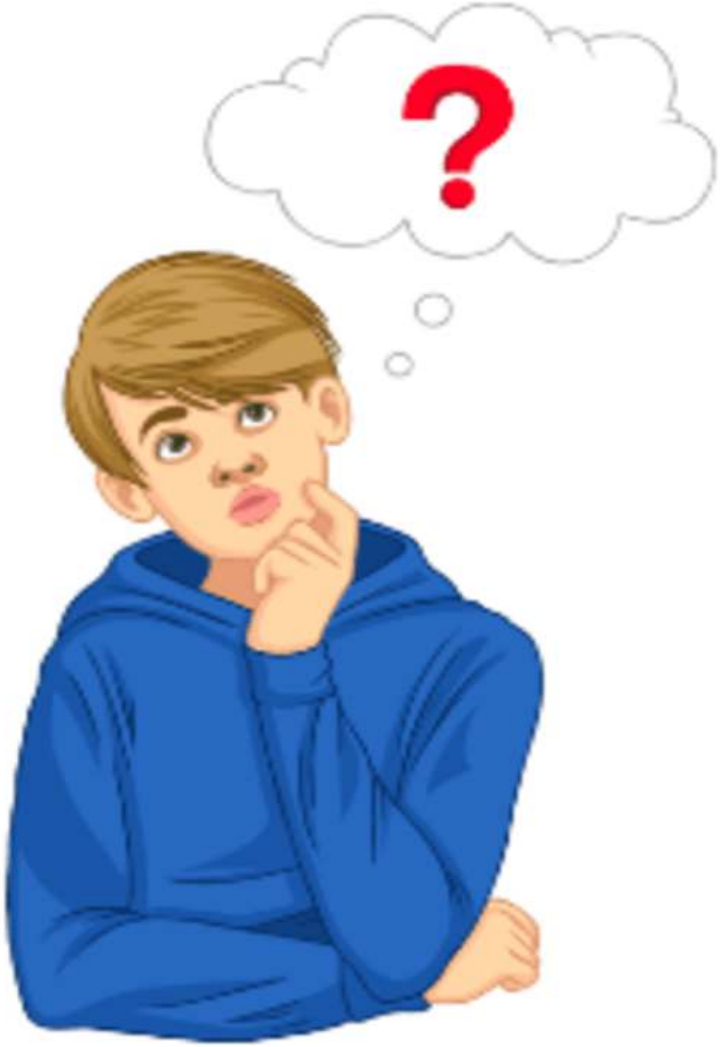
- Peep
- Madam
- Level
- Pop
- Radar

Purpose

Manacher’s algorithm is used to find the longest palindromic substring in a given string.

Eg- ababcbaz

megffgekl



Now the Question arises is Manacher's algorithm is the only way to solve this problem ?

Brute Force/naive Approach:

First find all possible sub-strings using nested loops, this solution has O(n^2) where n is the length of the given string. Then for every substring, check if it is a palindrome or not in O(n) , so the total time complexity is (n^3).

Dynamic programming Approach :

It is quite difficult to understand in just one video because DP in itself a huge topic .
The time complexity of the Dynamic Programming based solution is O(n^2) and it requires O(n^2) extra space.

This solution can further optimized by the idea is to Fix each char of a string as a center and expand in both directions for longer palindromes and keep track of the longest palindrome seen so far.
It's solution take (n^2) time with O(1) extra space

But the Question is can we do better ?

Finally

The answer is -YES

Manacher’s Algorithm-O(n)

Manacher’s Algorithm

- Manacher's algorithm is used to find the longest palindromic substring in any given string. This algorithm is faster than the brute force/DP approach, as it exploits the idea of a palindrome happening inside another palindrome.

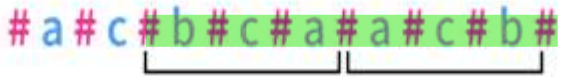
- Manacher's algorithm is designed to find the palindromic substrings with odd lengths only. To use it for even lengths also, we tweak the input string by inserting the character "#" at the beginning and each alternate position after that (changing "abcaac" to "#a#b#c#a#a#c#").

- In the case of an odd length palindrome, the middle character will be a character of the original string, surrounded by "#".

Odd length palindrome

Longest Palindrome

Even length palindrome



Longest Palindrome

s=abababa

Q=#a#b#a#b#a#b#a#.

p=[0, 1, 0, 3, 0, 5, 0, 7, 0, 5, 0, 3, 0, 1, 0]

Observation (1):

For the center index c=7 i.e p[c]=7 which has the longest palindromic substring. Notice that the numbers in array p after center c=7 are same as numbers before center c, so avoid expanding around all letters after center c, however just put their values directly using the Mirror (by copying the first half of array p in its other half) property.

Observation (2):

Unfortunately, Mirror property can't be applied in all cases. For example: s = "acncacn", the new string

Q= "#a#c#n#c#a#c#n#".

The result array p= [0 ,1 , 0 ,1 ,0 ,5 ,0 , 1 , 0 , 5 , 0 , 1 , 0 , 1 ,0].

So Mirror property doesn't work in all cases, because in this case there is another palindrome with center c=9. This new palindrome, with center c=9, goes beyond the limits of the first palindrome with center c=5.

Algorithm Steps:

Let the 2 limits of the first palindrome with center c: a left limit l, a right limit r. l ,r have references over the last 2 corresponding letters in the palindrome sub-string. A letter w with index i in a palindrome substring has a corresponding letter w' with index i' such that the c-i = i'-c.

(1) If(p[i]<=(r-i'),

So p[i']=p[i] which means that palindrome with center can't go beyond the original palindrome, so apply the Mirror Property directly.

(2) Else p[i']>=p[i],

This means that palindrome with center i’ goes beyond the original palindrome, so expanding around this center i’ is needed.

Let d=r-i , so expanding around center i’ will start from (i’-d)-1 with (i’+d)+1 and so on... because the interval[i’-d : i’+d] is already contained in the palindrome with center i’ .

The algorithm has 2 nested loops, outer loop check if there will be an expanding around current letter or not. This loop takes n steps.
Inner loop will be used in case of expanding around a letter, but it is guaranteed that it takes at most n steps by using the above 2 observations.
So the total time =2*n=O(n) .

3. Update c,r when a palindrome with center i' goes beyond the original palindrome with center c .
c=i' , r=i'+p[i'] as the next expanding will be around center .

MA_imp.cpp > main()

```cpp
// C++ solution for Longest Palindromic Substring (Manacher's Algorithm)
#include <bits/stdc++.h>
using namespace std;
#define SIZE 100000 + 1

int P[SIZE * 2];

// Transform S into new string with special characters inserted.
string convertToNewString(const string &s) {
    string newString = "@";

    for (int i = 0; i < s.size(); i++) {
        newString += "#" + s.substr(i, 1);
    }

    newString += "#$";
    return newString;
}
```

```
20  string longestPalindromeSubstring(const string &s) {
21      string Q = convertToNewString(s);
22      int c = 0, r = 0;                   // current center, right limit
23
24      for (int i = 1; i < Q.size() - 1; i++) {
25          // find the corresponding letter in the palidrome subString
26          int iMirror = c - (i - c);
27
28          if(r > i) {
29              P[i] = min(r - i, P[iMirror]);
30          }
31
32          // expanding around center i
33          while (Q[i + 1 + P[i]] == Q[i - 1 - P[i]]){
34              P[i]++;
35          }
36
37          // Update c,r in case if the palindrome centered at i expands past r,
38          if (i + P[i] > r) {
39              c = i;                  // next center = i
40              r = i + P[i];
41          }
42      }
```

```
43
44      // Find the longest palindrome length in p.
45
46      int maxPalindrome = 0;
47      int centerIndex = 0;
48
49      for (int i = 1; i < Q.size() - 1; i++) {
50
51          if (P[i] > maxPalindrome) {
52              maxPalindrome = P[i];
53              centerIndex = i;
54          }
55      }
56
57      cout << maxPalindrome << "\n";
58      return s.substr( (centerIndex - 1 - maxPalindrome) / 2, maxPalindrome);
59  }
60
61  int main() {
62      string s = "abababaca\n";
63      cout << longestPalindromeSubstring(s);
64      return 0;
65  }
```

Conclusion

Time Complexity of Manacher's algorithm is O(N)

Space Complexity of Manacher's algorithm is O(N)

Thank You