- Home
- Algorithms
- DS
- GATE
- Interview Corner
- Q&A
- C
- C++
- Java
- Books
- Contribute
- Ask a Q
- About

Array
Bit Magic
C/C++
Articles
GFacts
Linked List
MCQ
Misc
Output
String
Tree
Graph

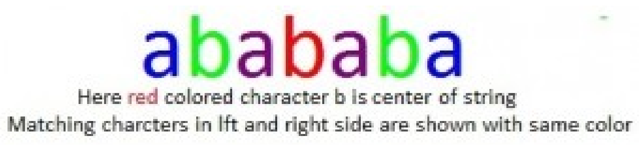
Manacher’s Algorithm – Linear Time Longest Palindromic Substring – Part 1

Given a string, find the longest substring which is palindrome.

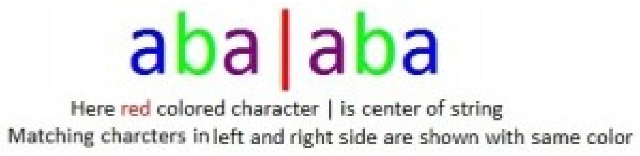
- if the given string is “forgeeksskeegfor”, the output should be “geeksskeeg”
- if the given string is “abaaba”, the output should be “abaaba”
- if the given string is “abababa”, the output should be “abababa”
- if the given string is “abcbabcbabcba”, the output should be “abcbabcba”

We have already discussed Naïve [O(n^3)] and quadratic [O(n^2)] approaches at Set 1 and Set 2.
In this article, we will talk about Manacher’s algorithm which finds Longest Palindromic Substring in linear time.

One way (Set 2) to find a palindrome is to start from the center of the string and compare characters in both directions one by one. If corresponding characters on both sides (left and right of the center) match, then they will make a palindrome.
Let's consider string "abababa".
Here center of the string is 4th character (with index 3) b. If we match characters in left and right of the center, all characters match and so string "abababa" is a palindrome.

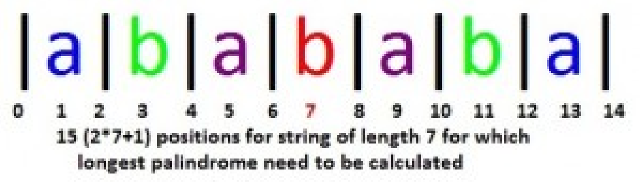

abababa
Here red colored character b is center of string
Matching charcters in lft and right side are shown with same color

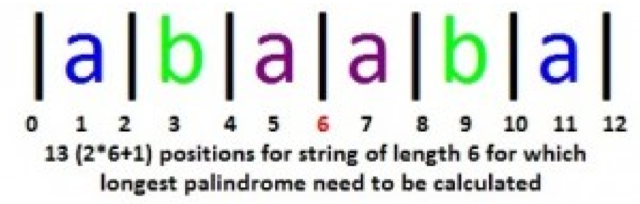
Here center position is not only the actual string character position but it could be the position between two characters also.
Consider string "abaaba" of even length. This string is palindrome around the position between 3rd and 4th characters a and a respectively.


aba|aba
Here red colored character | is center of string
Matching charcters in left and right side are shown with same color

To find Longest Palindromic Substring of a string of length N, one way is take each possible 2*N + 1 centers (the N character positions, N-1 between two character positions and 2 positions at left and right ends), do the character match in both left and right directions at each 2*N+ 1 centers and keep track of LPS. This approach takes O(N^2) time and that's what we are doing in Set 2.

Let's consider two strings "abababa" and "abaaba" as shown below:


|a|b|a|b|a|b|a|
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
15 (2*7+1) positions for string of length 7 for which longest palindrome need to be calculated


|a|b|a|a|b|a|
0 1 2 3 4 5 6 7 8 9 10 11 12
13 (2*6+1) positions for string of length 6 for which longest palindrome need to be calculated

In these two strings, left and right side of the center positions (position 7 in 1st string and position 6 in 2nd string) are symmetric. Why? Because the whole string is palindrome around the center position.
If we need to calculate Longest Palindromic Substring at each 2*N+1 positions from left to right, then palindrome's symmetric property could help to avoid some of the unnecessary computations (i.e. character comparison). If there is a palindrome of some length L cantered at any position P, then we may not need to compare all characters in left and right side at position P+1. We already calculated LPS at positions before P and they can help to avoid some of the comparisons after position P.
This use of information from previous positions at a later point of time makes the Manacher's algorithm linear. In Set 2, there is no reuse of previous information and so that is quadratic.

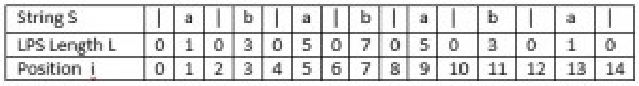
Manacher’s algorithm is probably considered complex to understand, so here we will discuss it in as detailed way as we can. Some of it’s portions may require multiple reading to understand it properly.

Let’s look at string “abababa”. In 3rd figure above, 15 center positions are shown. We need to calculate length of longest palindromic string at each of these positions.

- At position 0, there is no LPS at all (no character on left side to compare), so length of LPS will be 0.
- At position 1, LPS is a, so length of LPS will be 1.
- At position 2, there is no LPS at all (left and right characters a and b don’t match), so length of LPS will be 0.
- At position 3, LPS is aba, so length of LPS will be 3.
- At position 4, there is no LPS at all (left and right characters b and a don’t match), so length of LPS will be 0.
- At position 5, LPS is ababa, so length of LPS will be 5.

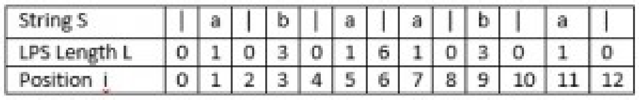
…… and so on

We store all these palindromic lengths in an array, say L. Then string S and LPS Length L look like below:



String S	|	a	|	b	|	a	|	b	|	a	|	b	|	a	|
LPS Length L	0	1	0	3	0	5	0	7	0	5	0	3	0	1	0
Position i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Similarly, LPS Length L of string “abaaba” will look like:



String S	|	a	|	b	|	a	|	a	|	b	|	a	|
LPS Length L	0	1	0	3	0	1	6	1	0	3	0	1	0
Position i	0	1	2	3	4	5	6	7	8	9	10	11	12

In LPS Array L:

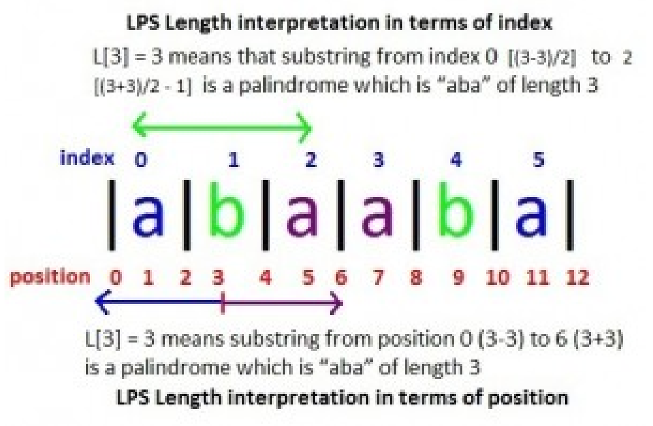
- LPS length value at odd positions (the actual character positions) will be odd and greater than or equal to 1 (1 will come from the center character itself if nothing else matches in left and right side of it)
- LPS length value at even positions (the positions between two characters, extreme left and right positions) will be even and greater than or equal to 0 (0 will come when there is no match in left and right side)

Position and index for the string are two different things here. For a given string S of length N, indexes will be from 0 to N-1 (total N indexes) and positions will be from 0 to 2*N (total 2*N+1 positions).

LPS length value can be interpreted in two ways, one in terms of index and second in terms of position. LPS value d at position I (L[i] = d) tells that:

- Substring from position i-d to i+d is a palindrome of length d (in terms of position)
- Substring from index (i-d)/2 to [(i+d)/2 – 1] is a palindrome of length d (in terms of index)

e.g. in string “abaaba”, L[3] = 3 means substring from position 0 (3-3) to 6 (3+3) is a palindrome which is “aba” of length 3, it also means that substring from index 0 [(3-3)/2] to 2 [(3+3)/2 – 1] is a palindrome which is “aba” of length 3.


LPS Length interpretation in terms of index
L[3] = 3 means that substring from index 0 [(3-3)/2] to 2 [(3+3)/2 - 1] is a palindrome which is "aba" of length 3
index 0 1 2 3 4 5
|a|b|a|a|b|a|
position 0 1 2 3 4 5 6 7 8 9 10 11 12
L[3] = 3 means substring from position 0 (3-3) to 6 (3+3) is a palindrome which is "aba" of length 3
LPS Length interpretation in terms of position

Now the main task is to compute LPS array efficiently. Once this array is computed, LPS of string S will be centered at position with maximum LPS length value.
We will see it in Part 2.

This article is contributed by **Anurag Singh**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Related Topics:

- Recursively print all sentences that can be formed from list of word lists
- Check if a given sequence of moves for a robot is circular or not
- Find the longest substring with k unique characters in a given string
- Function to find Number of customers who could not get a computer
- Find maximum depth of nested parenthesis in a string
- Find all distinct palindromic sub-strings of a given string
- Find if a given string can be represented from a substring by iterating the substring “n” times
- Suffix Tree Application 6 – Longest Palindromic Substring

Tweet 8+1 1

Writing code in comment? Please use **ideone.com** and share the link here.

5 Comments GeeksforGeeks 1 Login

Recommend Share Sort by Newest

Join the discussion…

Aashish Karki • 5 months ago
Palindrome using recursion: http://goo.gl/4mRQsA
• Reply • Share ›

Ankit Tripathi • 5 months ago
I think LPS w.r.t to Index should be: (i-d)/2 to (i+d)/2-1.
Because in the above example (3+3)/2 = 2, which is incorrect it should be 3.
Anyhow, great explanation till now. I was waiting for a clear explanation of this algorithm.
• Reply • Share ›

Anurag Singh Ankit Tripathi • 5 months ago
Thanks. It's corrected.
Next Part 2 is published.
• Reply • Share ›

geekyprateek • 5 months ago
Till now very exciting and clear expalination!! eagerly waiting for next part!!
1 • Reply • Share ›

Anurag Singh geekyprateek • 5 months ago
Next Part 2 is published.
• Reply • Share ›

Subscribe Add Disqus to your site Privacy DISQUS

Google™ Custom Search

-
-
-
-
  - Interview Experiences
  - Advanced Data Structures

- Dynamic Programming
- Greedy Algorithms
- Backtracking
- Pattern Searching
- Divide & Conquer
- Mathematical Algorithms
- Recursion
- Geometric Algorithms

Popular Posts

- All permutations of a given string
- Memory Layout of C Programs
- Understanding “extern” keyword in C
- Median of two sorted arrays
- Tree traversal without recursion and without stack!
- Structure Member Alignment, Padding and Data Packing
- Intersection point of two Linked Lists
- Lowest Common Ancestor in a BST.
- Check if a binary tree is BST or not
- Sorted Linked List to Balanced BST

Follow @GeeksforGeeks

Recent Comments

- Ashish Aggarwal

  Try Data Structures and Algorithms Made Easy -...

  Algorithms · 17 minutes ago

- Vlad

  Thanks. Very interesting lectures.

  Expected Number of Trials until Success · 1 hour ago

- cfh

  My implementation which prints the index of the...

  Longest Even Length Substring such that Sum of First and Second Half is same · 1 hour ago

- Gaurav pruthi

  forgot to see that part ;)

  Bloomberg Interview | Set 1 (Phone Interview) · 1 hour ago

- saeid aslami

  thanks

  Greedy Algorithms | Set 7 (Dijkstra’s shortest path algorithm) · 1 hour ago

- Cracker

  Implementation:...

  Implement Stack using Queues · 2 hours ago

-