GeeksforGeeks

A computer science portal for geeks

GeeksQuiz

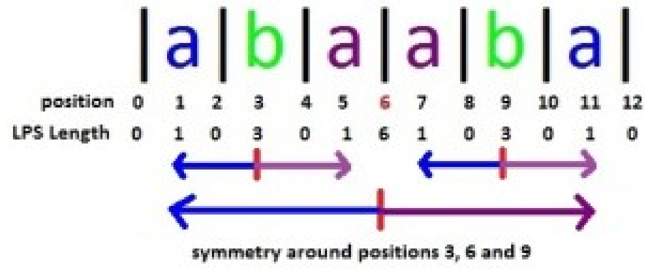
- Home
- Algorithms
- DS
- GATE
- Interview Corner
- Q&A
- C
- C++
- Java
- Books
- Contribute
- Ask a Q
- About

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

Manacher's Algorithm – Linear Time Longest Palindromic Substring – Part 2

In Manacher's Algorithm – Part 1, we gone through some of the basics and LPS length array.
Here we will see how to calculate LPS length array efficiently.

To calculate LPS array efficiently, we need to understand how LPS length for any position may relate to LPS length value of any previous already calculated position.
For string "abaaba", we see following:



If we look around position 3:

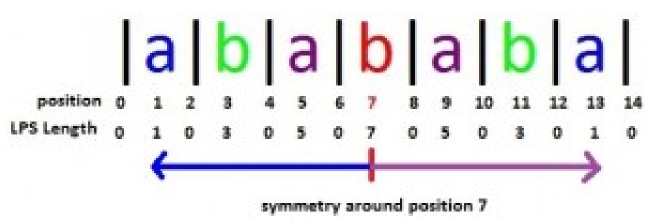
- LPS length value at position 2 and position 4 are same
- LPS length value at position 1 and position 5 are same

We calculate LPS length values from left to right starting from position 0, so we can see if we already know LPS length values at positions 1, 2 and 3 already then we may not need to calculate LPS length at positions 4 and 5 because they are equal to LPS length values at corresponding positions on left side of position 3.

If we look around position 6:

- LPS length value at position 5 and position 7 are same
- LPS length value at position 4 and position 8 are same

…………. and so on.
If we already know LPS length values at positions 1, 2, 3, 4, 5 and 6 already then we may not need to calculate LPS length at positions 7, 8, 9, 10 and 11 because they are equal to LPS length values at corresponding positions on left side of position 6.
For string "abababa", we see following:



If we already know LPS length values at positions 1, 2, 3, 4, 5, 6 and 7 already then we may not need to calculate LPS length at positions 8, 9, 10, 11, 12 and 13 because they are equal to LPS length values at corresponding positions on left side of position 7.

Can you see why LPS length values are symmetric around positions 3, 6, 9 in string "abaaba"? That's because there is a palindromic substring around these positions. Same is the case in string "abababa" around position 7.
Is it always true that LPS length values around at palindromic center position are always symmetric (same)?
Answer is NO.
Look at positions 3 and 11 in string "abababa". Both positions have LPS length 3. Immediate left and right positions are symmetric (with value 0), but not the next one. Positions 1 and 5 (around position 3) are not symmetric. Similarly, positions 9 and 13 (around position 11) are not symmetric.

At this point, we can see that if there is a palindrome in a string centered at some position, then LPS length values around the center position may or may not be symmetric depending on some situation. If we can identify the situation when left and right positions WILL BE SYMMETRIC around the center position, we NEED NOT calculate LPS length of the right position because it will be exactly same as LPS value of corresponding position on the left side which is already known. And this fact where we are avoiding LPS length computation at few positions makes Manacher's Algorithm linear.

In situations when left and right positions WILL NOT BE SYMMETRIC around the center position, we compare characters in left and right side to find palindrome, but here also algorithm tries to avoid certain no of comparisons. We will see all these scenarios soon.

Let's introduce few terms to proceed further:

(click to see it clearly)

- **centerPosition** – This is the position for which LPS length is calculated and let's say LPS length at centerPosition is d (i.e. L[centerPosition] = d)
- **centerRightPosition** – This is the position which is right to the centerPosition and d position away from centerPosition (i.e. **centerRightPosition = centerPosition + d**)
- **centerLeftPosition** – This is the position which is left to the centerPosition and d position away from centerPosition (i.e. **centerLeftPosition = centerPosition – d**)
- **currentRightPosition** – This is the position which is right of the centerPosition for which LPS length is not yet known and has to be calculated
- **currentLeftPosition** – This is the position on the left side of centerPosition which corresponds to the currentRightPosition
  centerPosition – currentLeftPosition = currentRightPosition – centerPosition
  currentLeftPosition = 2* centerPosition – currentRightPosition
- **i-left palindrome** – The palindrome i positions left of centerPosition, i.e. at currentLeftPosition
- **i-right palindrome** – The palindrome i positions right of centerPosition, i.e. at currentRightPosition
- **center palindrome** – The palindrome at centerPosition

When we are at centerPosition for which LPS length is known, then we also know LPS length of all positions smaller than centerPosition. Let's say LPS length at centerPosition is d, i.e. L[centerPosition] = d

It means that substring between positions "centerPosition-d" to "centerPosition+d" is a palindrom.
Now we proceed further to calculate LPS length of positions greater than centerPosition.
Let's say we are at currentRightPosition (> centerPosition) where we need to find LPS length.
For this we look at LPS length of currentLeftPosition which is already calculated.

If LPS length of currentLeftPosition is less than "centerRightPosition – currentRightPosition", then LPS length of currentRightPosition will be equal to LPS length of currentLeftPosition. So L[currentRightPosition] = L[currentLeftPosition] if L[currentLeftPosition] < centerRightPosition – currentRightPosition. This is **Case 1**.

Let's consider below scenario for string "abababa":

(click to see it clearly)

We have calculated LPS length up-to position 7 where L[7] = 7, if we consider position 7 as centerPosition, then centerLeftPosition will be 0 and centerRightPosition will be 14.
Now we need to calculate LPS length of other positions on the right of centerPosition.

For currentRightPosition = 8, currentLeftPosition is 6 and L[currentLeftPosition] = 0
Also centerRightPosition – currentRightPosition = 14 – 8 = 6
Case 1 applies here and so L[currentRightPosition] = L[8] = 0
Case 1 applies to positions 10 and 12, so,
L[10] = L[4] = 0
L[12] = L[2] = 0

If we look at position 9, then:
currentRightPosition = 9
currentLeftPosition = 2* centerPosition – currentRightPosition = 2*7 – 9 = 5
centerRightPosition – currentRightPosition = 14 – 9 = 5

Here L[currentLeftPosition] = centerRightPosition – currentRightPosition, so Case 1 doesn't apply here. Also note that centerRightPosition is the extreme end position of the string. That means center palindrome is suffix of input string. In that case, L[currentRightPosition] = L[currentLeftPosition]. This is **Case 2**.

Case 2 applies to positions 9, 11, 13 and 14, so:
L[9] = L[5] = 5
L[11] = L[3] = 3
L[13] = L[1] = 1
L[14] = L[0] = 0

What is really happening in Case 1 and Case 2? This is just utilizing the palindromic symmetric property and without any character match, it is finding LPS length of new positions.

When a bigger length palindrome contains a smaller length palindrome centered at left side of it's own center, then based on symmetric property, there will be another same smaller palindrome centered on the right of bigger palindrome center. If left side smaller palindrome is not prefix of bigger palindrome, then **Case 1** applies and if it is a prefix AND bigger palindrome is suffix of the input string itself, then **Case 2** applies.

*The longest palindrome i places to the right of the current center (the i-right palindrome) is as long as the longest palindrome i places to the left of the current center (the i-left palindrome) if the i-left palindrome is completely contained in the longest palindrome around the current center (the center palindrome) and the i-left palindrome is not a prefix of the center palindrome (**Case 1**) or (i.e. when i-left palindrome is a prefix of center palindrome) if the center palindrome is a suffix of the entire string (**Case 2**).*

In Case 1 and Case 2, i-right palindrome can't expand more than corresponding i-left palindrome (can you visualize why it can't expand more?), and so LPS length of i-right palindrome is exactly same as LPS length of i-left palindrome.

Here both i-left and i-right palindromes are completely contained in center palindrome (i.e. L[currentLeftPosition] <= centerRightPosition – currentRightPosition)
Now if i-left palindrome is not a prefix of center palindrome (L[currentLeftPosition] < centerRightPosition – currentRightPosition), that means that i-left palindrome was not able to expand up-to position centerLeftPosition.

If we look at following with centerPosition = 11, then

(click to see it clearly)

<table>
<tr><td>String S</td><td>|</td><td>c</td><td>|</td><td>d</td><td>|</td><td>b</td><td>|</td><td>a</td><td>|</td><td>b</td><td>|</td><td>c</td><td>|</td><td>b</td><td>|</td><td>a</td><td>|</td><td>b</td><td>|</td><td>d</td><td>|</td><td>b</td><td>|</td><td>a</td><td>|</td><td>b</td><td>|</td></tr>
<tr><td>LPS Length L</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>3</td><td>0</td><td>1</td><td>0</td><td>9</td><td>0</td><td>1</td><td>0</td><td>3</td><td>0</td><td>1</td><td>0</td><td>7</td><td>0</td><td>1</td><td>0</td><td>3</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>Position i</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td></tr>
</table>

centerLeftPosition would be 11 – 9 = 2, and centerRightPosition would be 11 + 9 = 20
If we take currentRightPosition = 15, it's currentLeftPosition is 7. Case 1 applies here and so L[15] = 3. i-left palindrome at position 7 is "bab" which is completely contained in center palindrome at position 11 (which is "dbabcbabd"). We can see that i-right palindrome (at position 15) can't expand more than i-left palindrome (at position 7).

If there was a possibility of expansion, i-left palindrome could have expanded itself more already. But there is no such possibility as i-left palindrome is prefix of center palindrome. So due to symmetry property, i-right palindrome will be exactly same as i-left palindrome and it can't expand more. This makes L[currentRightPosition] = L[currentLeftPosition] in Case 1.

Now if we consider centerPosition = 19, then centerLeftPosition = 12 and centerRightPosition = 26
If we take currentRightPosition = 23, it's currentLeftPosition is 15. Case 2 applies here and so L[23] = 3. i-left palindrome at position 15 is "bab" which is completely contained in center palindrome at position 19 (which is "babdbab"). In Case 2, where i-left palindrome is prefix of center palindrome, i-right palindrome can't expand more than length of i-left palindrome because center palindrome is suffix of input string so there are no more character left to compare and expand. This makes L[currentRightPosition] = L[currentLeftPosition] in Case 2.

Case 1: L[currentRightPosition] = L[currentLeftPosition] applies when:

- i-left palindrome is completely contained in center palindrome
- i-left palindrome is NOT a prefix of center palindrome

Both above conditions are satisfied when
L[currentLeftPosition] < centerRightPosition – currentRightPosition

Case 2: L[currentRightPosition] = L[currentLeftPosition] applies when:

- i-left palindrome is prefix of center palindrome (means completely contained also)
- center palindrome is suffix of input string

Above conditions are satisfied when
L[currentLeftPosition] = centerRightPosition – currentRightPosition (For 1st condition) AND centerRightPosition = 2*N where N is input string length N (For 2nd condition).

Case 3: L[currentRightPosition] > = L[currentLeftPosition] applies when:

- i-left palindrome is prefix of center palindrome (and so i-left palindrome is completely contained in center palindrome)

- center palindrome is NOT suffix of input string

Above conditions are satisfied when
L[currentLeftPosition] = centerRightPosition – currentRightPosition (For 1st condition) AND centerRightPosition < 2*N where N is input string length N (For 2nd condition).
In this case, there is a possibility of i-right palindrome expansion and so length of i-right palindrome is at least as long as length of i-left palindrome.

Case 4: L[currentRightPosition] > = centerRightPosition – currentRightPosition applies when:

- i-left palindrome is NOT completely contained in center palindrome

Above condition is satisfied when
L[currentLeftPosition] > centerRightPosition – currentRightPosition
In this case, length of i-right palindrome is at least as long (centerRightPosition – currentRightPosition) and there is a possibility of i-right palindrome expansion.

In following figure,

(click to see it clearly)

String S	|	b	|	a	|	b	|	c	|	b	|	a	|	b	|	c	|	b	|	a	|	c	|	c	|	b	|	a	|
LPS Length L	0	1	0	3	0	1	0	7	0	1	0	9	0	1	0	5	0	1	0	1	0	1	2	1	0	1	0	1	0
Position i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28

If we take center position 7, then Case 3 applies at currentRightPosition 11 because i-left palindrome at currentLeftPosition 3 is a prefix of center palindrome and i-right palindrome is not suffix of input string, so here L[11] = 9, which is greater than i-left palindrome length L[3] = 3. In the case, it is guaranteed that L[11] will be at least 3, and so in implementation, we 1st set L[11] = 3 and then we try to expand it by comparing characters in left and right side starting from distance 4 (As up-to distance 3, it is already known that characters will match).

If we take center position 11, then Case 4 applies at currentRightPosition 15 because L[currentLeftPosition] = L[7] = 7 > centerRightPosition – currentRightPosition = 20 – 15 = 5. In the case, it is guaranteed that L[15] will be at least 5, and so in implementation, we 1st set L[15] = 5 and then we try to expand it by comparing characters in left and right side starting from distance 5 (As up-to distance 5, it is already known that characters will match).

Now one point left to discuss is, when we work at one center position and compute LPS lengths for different rightPositions, how to know that what would be next center position. We change centerPosition to currentRightPosition if palindrome centered at currentRightPosition expands beyond centerRightPosition.

Here we have seen four different cases on how LPS length of a position will depend on a previous position's LPS length.
In Part 3, we have discussed code implementation of it and also we have looked at these four cases in a different way and implement that too.

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 g+1 1

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

6 Comments GeeksforGeeks 1 Login

Recommend Share Sort by Newest

Join the discussion…

Rachit Sachdeva • 3 months ago

Please correct the following error.

"If we look at position 9, then:currentRightPosition = 9
currentLeftPosition = 2* centerPosition – currentRightPosition = 2*7 – 9 = 5

centerRightPosition – currentRightPosition = 14 – 9 = 5"

It should be "If we look at position *7*,... "

Reply • Share ›

Martin 'Vegi' Kysel • 4 months ago

Hey man, you are missing a link to the next part. I had to navigate via URL manipulation. Did I miss something? (I still don't get the algorithm, but as soon as I will I shall salute your work!)

Reply • Share ›

Anurag Singh → Martin 'Vegi' Kysel • 4 months ago

Thanks for pointing this. Link to next part is provided in the end of this article which is Manacher’s Algorithm Part 3

Reply • Share ›

geekyprateek • 5 months ago

Nyc work !! A more clear explaination is given here and with more clear terminologies

http://tarokuriyama.com/projec...

Reply • Share ›

• Reply • Share ›

Nishant • 5 months ago

Good work dude, but i think this is simple concept made complex using terminologies. Here is intuition of above algo.
Consider the string = "abababa"(see above dig. for position). At position 7('b'), we have palindrome of length 7.
Left portion of palindrome('aba') = Right portion of palindrome('aba').
case I: we want to compute palindrome length at position 10. Map this position to left half = 4. Now we know that at 4, length of palindrome is 0 which is "less than" size of left portion, and because left half and right half are exactly same, corresponding palindrome value at 10 also can not exceed right portion(which is same as left portion) and its value is equal to palindrome value at 4. This is "i-left palindrome IS completely contained in center palindrome"
Now consider string "baabababaxx", here we appended "ba" at beginning and "xx" at end. suppose we have computed values till position 11('b') which is =7. We want to compute value at position 17('a'), this correspond to position on left portion = 5('a') whose value is 5. Now we know that at position 5, left portion of it "ba" is matching right portion "ba", however for position 17, we don't know anything about its right side that is "xx". so we can't conclude anything. This correspond to "i-left palindrome NOT completely contained in center palindrome".
Similarly if "i-left palindrome IS a prefix of center palindrome" then corresponding right position may have palindrome value greater than its left position".

see more

2 • Reply • Share ›

Anurag Singh → Nishant • 5 months ago

You are right. There are concepts which can be thought and explained in different ways. Some may look simple, some may look complex. Manacher’s Algorithm is probably one of them.
If you are new and don't know how it works, you may find in hard to understand.
Once you spend some time and understand how it really works, you may find a better way to interpret it in your own way.
This article is intended to explain the concepts in detailed way and you see four different cases here which could have increased the complexity.
The statement "also we will look at these four cases in a different way" at the end of article meant what you said. These four cases can be seen in a simpler way and we will talk about that.

• 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

@geeksforgeeks, Some rights reserved Contact Us!
Powered by WordPress & MooTools, customized by geeksforgeeks team