Longest Palindromic Substring Part II

November 20, 2011 by 1337c0d3r 157 Replies

Given a string S, find the longest palindromic substring in S.

Note:

This is Part II of the article: Longest Palindromic Substring. Here, we describe an algorithm (Manacher's algorithm) which finds the longest palindromic substring in linear time. Please read Part I for more background information.

In my previous post we discussed a total of four different methods, among them there's a pretty simple algorithm with $O(N^2)$ run time and constant space complexity. Here, we discuss an algorithm that runs in O(N) time and O(N) space, also known as Manacher's algorithm.

Hint:

Think how you would improve over the simpler $O(N^2)$ approach. Consider the worst case scenarios. The worst case scenarios are the inputs with multiple palindromes overlapping each other. For example, the inputs: "aaaaaaaaa" and "cabcbabcbabcba". In fact, we could take advantage of the palindrome's symmetric property and avoid some of the unnecessary computations.

An O(N) Solution (Manacher's Algorithm):

First, we transform the input string, S, to another string T by inserting a special character '#' in between letters. The reason for doing so will be immediately clear to you soon.

For example: S = "abaaba", T = "#a#b#a#a#b#a#".

To find the longest palindromic substring, we need to expand around each T_i such that $T_{i-d} \ldots T_{i+d}$ forms a palindrome. You should immediately see that *d* is the length of the palindrome itself centered at T_i.

We store intermediate result in an array P, where P[i] equals to the length of the palindrome centers at T_i. The longest palindromic substring would then be the maximum element in P.

Using the above example, we populate P as below (from left to right):

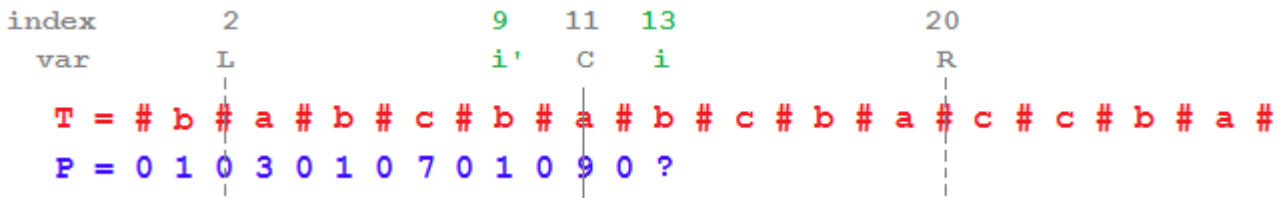
```
T = # a # b # a # a # b # a #
P = 0 1 0 3 0 1 6 1 0 3 0 1 0
```

Looking at P, we immediately see that the longest palindrome is “abaaba”, as indicated by $P_6 = 6$.

Did you notice by inserting special characters (#) in between letters, both palindromes of odd and even lengths are handled graciously? (Please note: This is to demonstrate the idea more easily and is not necessarily needed to code the algorithm.)

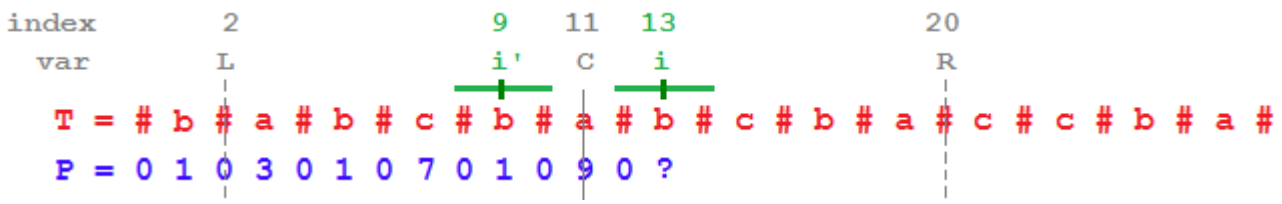
Now, imagine that you draw an imaginary vertical line at the center of the palindrome “abaaba”. Did you notice the numbers in P are symmetric around this center? That’s not only it, try another palindrome “aba”, the numbers also reflect similar symmetric property. Is this a coincidence? The answer is yes and no. This is only true subjected to a condition, but anyway, we have great progress, since we can eliminate recomputing part of P[i]’s.

Let us move on to a slightly more sophisticated example with more some overlapping palindromes, where S = “babcbabcbaccba”.



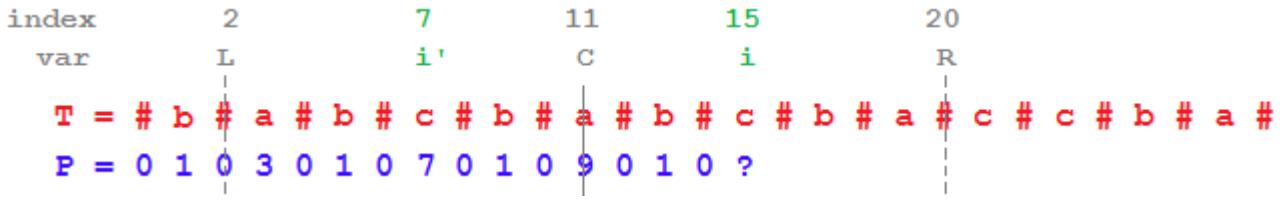
Above image shows T transformed from S = “babcbabcbaccba”. Assumed that you reached a state where table P is partially completed. The solid vertical line indicates the center (C) of the palindrome “abcbabcba”. The two dotted vertical line indicate its left (L) and right (R) edges respectively. You are at index i and its mirrored index around C is i’. How would you calculate P[i] efficiently?

Assume that we have arrived at index i = 13, and we need to calculate P[13] (indicated by the question mark ?). We first look at its mirrored index i’ around the palindrome’s center C, which is index i’ = 9.



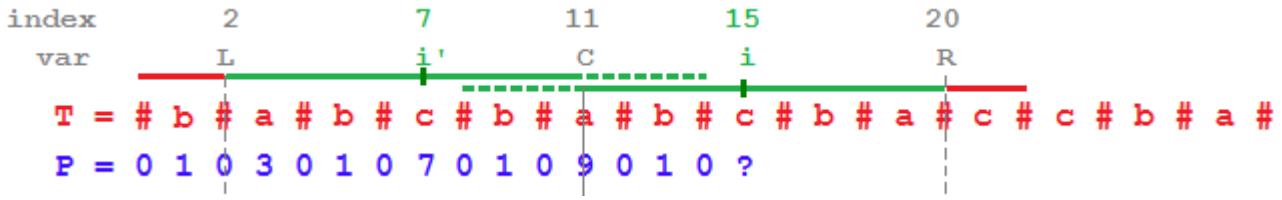
The two green solid lines above indicate the covered region by the two palindromes centered at i and i’. We look at the mirrored index of i around C, which is index i’. P[i’] = P[9] = 1. It is clear that P[i] must also be 1, due to the symmetric property of a palindrome around its center.

As you can see above, it is very obvious that P[i] = P[i'] = 1, which must be true due to the symmetric property around a palindrome's center. In fact, all three elements after C follow the symmetric property (that is, P[12] = P[10] = 0, P[13] = P[9] = 1, P[14] = P[8] = 0).



Now we are at index i = 15, and its mirrored index around C is i' = 7. Is P[15] = P[7] = 7?

Now we are at index i = 15. What's the value of P[i]? If we follow the symmetric property, the value of P[i] should be the same as P[i'] = 7. But this is wrong. If we expand around the center at T_{15}, it forms the palindrome "a#b#c#b#a", which is actually shorter than what is indicated by its symmetric counterpart. Why?



Colored lines are overlaid around the center at index i and i'. Solid green lines show the region that must match for both sides due to symmetric property around C. Solid red lines show the region that might not match for both sides. Dotted green lines show the region that crosses over the center.

It is clear that the two substrings in the region indicated by the two solid green lines must match exactly. Areas across the center (indicated by dotted green lines) must also be symmetric. Notice carefully that P[i '] is 7 and it expands all the way across the left edge (L) of the palindrome (indicated by the solid red lines), which does not fall under the symmetric property of the palindrome anymore. All we know is P[i] ≥ 5, and to find the real value of P[i] we have to do character matching by expanding past the right edge (R). In this case, since P[21] ≠ P[1], we conclude that P[i] = 5.

Let's summarize the key part of this algorithm as below:

> **if** P[i'] ≤ R – i,
> **then** P[i] ← P[i']
> **else** P[i] ≥ P[i']. (Which we have to expand past the right edge (R) to find P[i].

See how elegant it is? If you are able to grasp the above summary fully, you already obtained the essence of this algorithm, which is also the hardest part.

The final part is to determine when should we move the position of C together with R to the right, which is easy:

> If the palindrome centered at i does expand past R, we update C to i, (the center of this new palindrome), and extend R to the new palindrome's right edge.

In each step, there are two possibilities. If P[i] ≤ R – i, we set P[i] to P[i'] which takes exactly one step. Otherwise we attempt to change the palindrome's center to i by expanding it starting at the right edge, R. Extending R (the inner while loop) takes at most a total of N steps, and positioning and testing each centers take a total of N steps too. Therefore, this algorithm guarantees to finish in at most 2*N steps, giving a linear time solution.

```
// Transform S into T.
// For example, S = "abba", T = "^#a#b#b#a#$".
// ^ and $ signs are sentinels appended to each end to avoid bounds checking
string preProcess(string s) {
  int n = s.length();
  if (n == 0) return "^$";
  string ret = "^";
  for (int i = 0; i < n; i++)
    ret += "#" + s.substr(i, 1);

  ret += "#$";
  return ret;
}

string longestPalindrome(string s) {
  string T = preProcess(s);
  int n = T.length();
  int *P = new int[n];
  int C = 0, R = 0;
  for (int i = 1; i < n-1; i++) {
    int i_mirror = 2*C-i; // equals to i' = C - (i-C)

    P[i] = (R > i) ? min(R-i, P[i_mirror]) : 0;

    // Attempt to expand palindrome centered at i
    while (T[i + 1 + P[i]] == T[i - 1 - P[i]])
      P[i]++;

    // If palindrome centered at i expand past R,
    // adjust center based on expanded palindrome.
    if (i + P[i] > R) {
      C = i;
      R = i + P[i];
    }
  }

  // Find the maximum element in P.
  int maxLen = 0;
  int centerIndex = 0;
  for (int i = 1; i < n-1; i++) {
    if (P[i] > maxLen) {
      maxLen = P[i];
      centerIndex = i;
    }
  }
```

```
  delete[] P;

  return s.substr((centerIndex - 1 - maxLen)/2, maxLen);
}
```

Note:

This algorithm is definitely non-trivial and you won't be expected to come up with such algorithm during an interview setting. However, I do hope that you enjoy reading this article and hopefully it helps you in understanding this interesting algorithm. You deserve a pat if you have gone this far! 🙂

Further Thoughts:

- In fact, there exists a sixth solution to this problem — Using suffix trees. However, it is not as efficient as this one (run time O(N log N) and more overhead for building suffix trees) and is more complicated to implement. If you are interested, read Wikipedia's article about Longest Palindromic Substring.
- What if you are required to find the longest palindromic subsequence? (Do you know the difference between substring and subsequence?)

Useful Links:

» Manacher's Algorithm O(N) 时间求字符串的最长回文子串 (Best explanation if you can read Chinese)
» A simple linear time algorithm for finding longest palindrome sub-string
» Finding Palindromes
» Finding the Longest Palindromic Substring in Linear Time
» Wikipedia: Longest Palindromic Substring

Rating: 4.9/**5** (204 votes cast)

Longest Palindromic Substring Part II, 4.9 out of 5 based on 204 ratings

← Longest Palindromic Substring Part I Palindrome Number →

157 thoughts on "Longest Palindromic Substring Part II"

Pingback: Longest Palindromic Substring Part I | LeetCode

wayne
November 21, 2011 at 8:09 am

i think my solution is simpler than this one, the key point of my solution is:
The center point of palindromic substring is always follow this pattern, either is “…..XyX…..” or “….XX….”.

so you can scan once and then find those center point of palindromic substring and then expand it on each center points to find the one with maxium length.

i ve already posted my java solution in the comments of Longest Palindromic Substring Part I

Reply ↓ **-44**

1337c0d3r Post author
November 22, 2011 at 12:17 am

Yes your solution is simpler but runs in O(N^2) worst case. It is already discussed in my previous post.

Reply ↓ **+8**

wayne
November 22, 2011 at 7:07 am

i agreed, it is O(N^2) worst case, thanks.

Reply ↓ **-7**

1337c0d3r Post author
November 22, 2011 at 5:20 pm

No problem.
Basically this algorithm is an improvement over your method. It is using the symmetric property of a palindrome to eliminate some of the recomputations of palindrome’s length, and amazingly improve it to a linear time solution.

Reply ↓ **-1**

Andres
September 24, 2012 at 12:45 pm

Excellent post, I learnt a lot

Thanks!

-4

Reply ↓

GuojiaAgain
January 20, 2014 at 10:45 pm

Your implementation is simpler but cost more time.

Reply ↓ Report user -5

Sreekar
November 21, 2011 at 11:01 pm

Looks like this is O(N^2) algorithm as there is a while loop in for loop. Could you please clarify?

Reply ↓ -3

1337c0d3r Post author
November 22, 2011 at 12:16 am

Even with the extra while loop inside, it is guaranteed in the worst case the algorithm completes in 2*n steps.

Think of how the i and right edge (R) relates. In the loop each time, you look if this index is a candidate to re-position the palindrome's center. If it is, you increment the existing R one at a time. See? R could only be incremented at most N steps. Once you incremented a total of N steps, it couldn't be incremented any more. It's not like you will increment R all the time in the while loop. This is called amortized O(1).

Reply ↓ +9

Vikas
January 5, 2013 at 10:59 am

How is it amortised O(1) ? I am confused from the definition from amortise.

Amortised to me means , for a worst case of runs of some operations its amortised cost over all of them

Reply ↓ 0

yy
March 20, 2014 at 12:26 am

Great, I see why it's amortized O(1) now. Thanks!

Reply ↓ -2

Jason

September 16, 2014 at 12:48 am

It seems a O(N^2) alg. For example, "abcdcba", go through the string needs N step, and the while loop needs N/2 step when meets character 'd', so O (N*N/2) =O(N^2)?

Reply ↓ +1

Kimi

April 25, 2015 at 8:32 pm

Jason I think you misunderstood, only one step took N/2 steps, so you cannot assume the time taken is N/2 * N, for your case, should be N + (1 + 1 + 1…+ N/2), 1+1+1+… = N/2, so totally N+(N/2 + N/2) = 2N, so clearly O(N) for this alg.

Reply ↓ Report user +2

Sreekar

November 22, 2011 at 12:32 am

Got it. Thanks for the clarification. Great solution

Reply ↓ +2

1337c0d3r Post author

November 22, 2011 at 5:19 pm

Thanks! Hope you understands it. Let me know if you have any more questions!

Reply ↓ +1

coderrush

February 29, 2012 at 12:43 pm

I am wondering if the algorithm is O(NlgN) or O(N)?

+2

Reply ↓

flo
November 22, 2011 at 4:51 am

Great write-up. Thanks for the article

Reply ↓ Report user +2

1337c0d3r Post author
November 22, 2011 at 5:14 pm

Thanks!
My goal of writing this article is to provide an intuitive way to understand the algorithm. I hope you really appreciate the beauty of this algorithm.

Reply ↓ +8

J
May 5, 2012 at 9:25 pm

I would if I could understand it.

Reply ↓ +3

vaibhav
November 24, 2011 at 1:03 am

while explaining how to fill P[i] you mentioned
”
if P[i’] ≤ R – i,
then P[i] ← P[i’]
else P[i] ≥ P[i’]. (Which we have to expand past the right edge (R) to find P[i].
”

is the else statement right?? shouldnt be “else P[i’] ≥ P[i]”

Reply ↓ +15

ironmanz
May 17, 2013 at 2:12 pm

I agree with you.

Reply ↓ Report user 0

YN
May 23, 2015 at 7:11 pm

should be:
“else P[i’] ≥ R-i”

Reply ↓ +9

k2516
October 8, 2015 at 10:13 pm

should be:
“else P[i]>=R-i”

Reply ↓ +5

bleu
November 24, 2011 at 4:59 am

It should rather be:
else P[i] ≥ (R-i) (Which we have to expand past the right edge (R) to find P[i])
.. Also note how coherent the reasoning in the bracket sounds now.

Explanation :
When P[i’] > R-i then all we know, by symmetry about C, is :
P[i’] > R-i .. by obviousness
and
P[i] ≥ R-i .. by the meaning of R
From this we clearly cannot conclude upon max(P[i’], P[i])

Reply ↓ +9

J
May 5, 2012 at 9:24 pm

Don’t see the coherency.

Reply ↓

ClaireQu
September 17, 2014 at 7:14 pm

I agree with you

Reply ↓ Report user 0

Lei
February 17, 2015 at 2:03 am

I agree

Reply ↓ 0

Amit
June 24, 2016 at 7:17 am

I agree.

Reply ↓ 0

Fei
November 25, 2011 at 2:56 pm

Using suffix tree can do this in O(n). And building suffix tree can be also done in O(n): http://blog.csdn.net/g9yuayon/article/details/2574781

But this algorithm is pretty cool too!

Reply ↓ +2

1337c0d3r Post author
November 29, 2011 at 12:27 pm

Thanks Fei! I will look into that.

Reply ↓ 0

LB
November 29, 2011 at 12:05 pm

Clear explanation. It could hardly be any better 🙂

Thumbs up for elucidating this magic O(n) solution in such intuitive manner. You got talent to clearly expressing an algorithm, which I even missed in books like Cormen's Algorithm!

Reply ↓ +2

1337c0d3r Post author
November 29, 2011 at 12:24 pm

Thanks! Good to know I've done my job — to introduce tricky but interesting algorithms in an intuitive manner.

Reply ↓ +2

Bahlul Haider
December 1, 2011 at 12:28 pm

Really beautiful algorithm.

Reply ↓ Report user +1

Googmeister
December 6, 2011 at 2:11 am

Wonderful writeup with great illustrations! I think there is one minor bug in your code: if s itself is a palindrome, then the following line accesses the array out of bounds.

```
    while (T[i + 1 + P[i]] == T[i - 1 - P[i]])
```

Reply ↓ +2

1337c0d3r Post author
December 6, 2011 at 11:15 am

ahh… you are right! Thanks for your sharp observation.

I thought that I feel something is not right when I decided to add '$' both to the begin and the end of the input

string. (It should be adding two different sentinels '^' and '$' to the begin and the end of the string. This should avoid bounds checking and the out of bounds problem)

Reply ↓ +2

online chesstle
February 15, 2014 at 10:16 am

s.substr((centerIndex – 1 – maxLen)/2, maxLen); is there a bug here as the rest of the code is treating P[i] as the length of the palindrome on either side of the center and not as the total length of the palindrome.

Reply ↓ 0

online chesstle
February 15, 2014 at 10:28 am

a fault in the code here? s.substr((centerIndex – 1 – maxLen)/2, maxLen);

the core algortihm is handling P[i] as the length of the palindrome on either side of the center and not as the total length of the palindrome.

Reply ↓ 0

online chesstle
February 15, 2014 at 10:31 am

nevermind, accounting for the hashes.

Reply ↓ +2

Karthick
December 12, 2011 at 5:15 am

Thanks for such a lucid explanation. I had already visited all the references that you had suggested at the end. I was not able to understand the essence of it until I read yours. 🙂

Well, I have one question.
Is it possible to run the algorithm without using the '#','^','$' symbols?

Reply ↓ +1

Karthick
December 12, 2011 at 7:28 am

As an after-thought, I have one doubt.
Why are we using the line
P[i] = (R > i) ? min(R-i, P[i_mirror]) : 0;

Can you please clarify this?

Reply ↓ Report user 0

Pingback: Palindrome Number | LeetCode

MSJ
January 22, 2012 at 9:16 am

there are another two solutions
1) suffix array version preprocess requires N*logN query is O(N) http://www.mashengjun.info/?p=901
2) another O(N) solution http://www.mashengjun.info/?p=464 using up&down pointer

Reply ↓ Report user 0

1337c0d3r Post author
January 22, 2012 at 3:09 pm

Did you try running your code through Online Judge? http://www.leetcode.com/onlinejudge
It did not pass all test cases.

Reply ↓ 0

caopeng
March 2, 2012 at 6:53 am

The seconde is not O(N) it's O(N^2) i think…

Reply ↓ 0

Caopeng
March 1, 2012 at 11:30 pm

The first i_mirror is -1 which is less than 0 so there may be run time error?

Reply ↓ Report user +2

Anonymous
March 19, 2012 at 6:02 am

```
while (T[i + 1 + P[i]] == T[i - 1 - P[i]])
P[i]++;
```

It's really O(N)?

Reply ↓ +1

Vyas
April 9, 2012 at 11:41 am

A Very nice explanation!!!!:)
One thing I could not understand… "In this case, since P[21] ≠ P[1], we conclude that P[i] = 5.".. In this statement, from what I have understood, I think it should be P[21] ≠ P[8]. Please correct me if I'm wrong…..

Reply ↓ +1

Kareem
May 1, 2012 at 12:52 am

The conclusion of the algorithm above states that
if P[i'] ≤ R – i,
then P[i] ← P[i']
else P[i] ≥ P[i']. (Which we have to expand past the right edge (R) to find P[i].

The first check should be P[i'] < R – i, as when they are equal the proper value of p[i] can not be fully determined with P[i'] only but needs to expand.
for example: string #b#b#a#b#a#b#a# with i = 9, c = 7, R = 12

Reply ↓ +1

Li
August 2, 2012 at 11:51 am

Agree with you.

0

Reply ↓

J
May 5, 2012 at 8:12 pm

It is clear that the two substrings in the region indicated by the two solid green lines must match exactly. Areas across the center (indicated by dotted green lines) must also be symmetric.

i is in green area (index 15). So I should have the same value as i[7]. but it doesn't.

So what is going on?

Reply ↓ **0**

J
May 5, 2012 at 8:13 pm

The first two lines are quotes from the written explanation.

Reply ↓ **0**

Chev
May 27, 2012 at 10:19 pm

Is there a problem in the following code? i_mirror will be -1 and P[i_mirror] will be out of boundary when C=0 and i =1, or do I miss something? Thanks.

…
int C = 0, R = 0;
for (int i = 1; i i) ? min(R-i, P[i_mirror]) : 0;
…
}

Reply ↓ **0**

Chev
May 27, 2012 at 11:03 pm

I see! It is my mistake. Thanks!

Reply ↓ **0**

Chev
May 27, 2012 at 10:21 pm

Here is the piece of code where I am confused:
int C = 0, R = 0;
for (int i = 1; i i) ? min(R-i, P[i_mirror]) : 0;

Reply ↓ +1

Chev
May 27, 2012 at 11:03 pm

Got it! I am clear now. Thanks!

Reply ↓ 0

Chuanyou Li
July 6, 2012 at 9:37 pm

Should P[i] R – i ??

Reply ↓ 0

Pingback: Longest Palindromic Substring « Interview Algorithms

Noone
September 5, 2012 at 7:48 am

Have to disagree with the others. You have actually managed to complicate a simple algorithm!

The key idea is quite simple actually.

Reply ↓ 0

Subramanian Ganapathy
October 14, 2012 at 4:17 pm

Recurrence:

L[i] = max{L[i-1] + 1, #Arr[i-1 – L[i-1]…i] is univalue.
L[i-1]+2 #Arr[i] == Arr[i-1-L[i-1]]}

L[i]=1 otherwise.

Am i missing something here? this recurrrence solves it and it is a lot simpler.

Reply ↓ +1

Subramanian Ganapathy
October 14, 2012 at 5:04 pm

My bad 🙂 my recurrence messes up the overlapping palindromes case, awesome solution and nice explanation. you deserve a pat on your back 🙂 thank you

Reply ↓ 0

Pingback: Longest Palindromic Substring » KEVIN'S BLOG

Karthick
November 1, 2012 at 11:04 pm

The following is the implementation of the Manacher's algorithm without pre-processing the input string. It is a bit clumsy – sorry for that. I tested it using the online judge here and it seems to be working fine.

Language : java

```
public String longestPalindrome(String s) {
    if(s==null)
        return "";

    int len=s.length();
    int[] p=new int[2*len-1];
    p[0]=1;
    int R=0,C=0;
    int curLen,l,r,start;

    for(int i=1;ii) ? Math.min(curLen,p[iMirror]) : ( i%2==0 ? 1 : 0) );
        if(i%2==0){
            l=(i/2-p[i]/2-1);
            r=(i/2+p[i]/2+1);
        }
        else{
            l=(i/2-p[i]/2);
            r=(i/2+p[i]/2+1);
        }

        while(l>=0 && rR){
            C=i;
            R=r-1;
        }
```

```
        }

        int maxIndex=getMaxIndex(p);
        if(maxIndex%2==0)
            start=(maxIndex/2-p[maxIndex]/2);
        else
            start=(maxIndex/2-p[maxIndex]/2 +1);

        return s.substring(start,start+p[maxIndex]);

    }

    int getMaxIndex(int p[]){
        int len=p.length;
        int maxIndex=0;
        for(int i=1;ip[maxIndex])
                maxIndex=i;
        }
        return maxIndex;
    }
```

Reply ↓ -2

Karthick
November 1, 2012 at 11:08 pm

Sorry,there seems to be some problem – some part of the code seems to be omitted when I post my code using the

```
```

tag. So, I am posting it without the code tag.

public String longestPalindrome(String s) {

if(s==null)
return “”;

int len=s.length();
int[] p=new int[2*len-1];
p[0]=1;
int R=0,C=0;
int curLen,l,r,start;
for(int i=1;ii) ? Math.min(curLen,p[iMirror]) : (i%2==0 ? 1 : 0));
if(i%2==0){
l=(i/2-p[i]/2-1);
r=(i/2+p[i]/2+1);

```
}
else{
l=(i/2-p[i]/2);
r=(i/2+p[i]/2+1);
}

while(l>=0 && rR){
C=i;
R=r-1;
}

}

int maxIndex=getMaxIndex(p);
if(maxIndex%2==0)
start=(maxIndex/2-p[maxIndex]/2);
else
start=(maxIndex/2-p[maxIndex]/2 +1);

return s.substring(start,start+p[maxIndex]);

}

int getMaxIndex(int p[]){
int len=p.length;
int maxIndex=0;
for(int i=1;ip[maxIndex])
maxIndex=i;
}
return maxIndex;
}
```

Reply ↓ **0**

abhay

January 5, 2013 at 12:08 pm

nice explaination thanks for article..

😛

Reply ↓ **0**

Naveen Kumar
January 16, 2013 at 4:28 am

I am stuck at summarized part of this algo.
if P[i'] ≤ R – i,
then P[i] ← P[i']
else P[i] ≥ P[i']. (Which we have to expand past the right edge (R) to find P[i].)
how could we say which one be large for else part.?
even in example for i=15,p[i]=5,i'=7 p[i']=7;
i m confused here. plz help me.
Thanks..

Reply ↓ +1

zed
October 19, 2015 at 9:20 pm

we can not say that, in the else case, we just have to expand R and recenter C

Reply ↓ 0

Mony Sim
January 20, 2013 at 12:18 pm

try simple solution.
bool is palindrome(string s)
{ int len=s.length(), a=0, b=len-1;
while(a<b)
{
if(s[a]!=s[b])
return false;
}
return true;
}

Reply ↓ Report user -5

Mony Sim
January 20, 2013 at 12:29 pm

try simple solution.

```
bool is palindrome(string s)
{ int len=s.length(), a=0, b=len-1;
while(a<b)
{
if(s[++a]!=s[–b])
return false;
}
return true;
}
```

Reply ↓ Report user -1

rajat rastogi
February 2, 2013 at 8:06 am

Dude your solution is O(n^2) take a case of a string aaaaaa, palindrome length is 6 and solution for this string confirms O(n^2)

Reply ↓ +1

William Gozali
April 14, 2013 at 3:11 am

I don’t think so.

Take a look at the explanation, it is guaranteed that the operation needed will not exceed O(N). Maybe you should try to simulate the algorithm with that input

Reply ↓ Report user -1

Žilvinas
February 4, 2013 at 3:50 am

Thanks so much, exelent tutorial!

Reply ↓ Report user 0

Nipun Poddar
February 6, 2013 at 8:07 am

really nice one..helpedme to learn a lot.. 🙂

Reply ↓ 0

Pingback: Longest Palindromic Substring » Kevin's Tech Blog

Kuldeep Yadav
March 15, 2013 at 9:18 pm

GOOD WORK 🙂

Reply ↓ Report user -1

Ayaskant Swain
April 25, 2013 at 12:01 am

Wonderful post. Thanks for posting these kind of problems and their solutions which will help in interviews. I have a question in this part-II solution.

I think the complexity will be still O(n * n), since you are traversing the string twice actually. One for modifying the original input string to insert characters ^,#,$ and then again you will do another traversal from the beginning to end of the string to search for actual palindromes in the string.

Reply ↓ Report user -1

lagrange
June 2, 2013 at 4:12 pm

我觉得那个中文的blog, p[i]表示向左/右延展的长度, 比p[i]表示整个substr的长度要更容易理解一些

Reply ↓ +1

zhuyao
September 2, 2014 at 6:57 am

aglee!

Reply ↓ -1

Pingback: leetcode: Longest Palindromic Substring solution | 烟客旅人 sigmainfy

shivali
June 17, 2013 at 9:25 pm

can you please xplain how you calculated the complexity of the above algorithm

Reply ↓ 0

shivali
June 18, 2013 at 6:44 am

```
//longest pallindrome in a string(c++)
#include
#include
#define lsfor(i,a,b) for(i=a;i<b;i++)

using namespace std;
char str[100];
int n,curr_len=0,max_len=1,l,r,t;

int main()
{
int i;
cout<<"enter no.of test cases:"<>t;
while(t–)
{
cout<<"please enter your string"<>str;
n=strlen(str);
if(n==1)
{cout<<"1"<<endl;
return(0);}

if(n==2)

{cout<<"2"<=0&&r<=n-1)
{
if(str[l]==str[r])
{
curr_len+=2;
if(max_len<curr_len)
max_len=curr_len;
```

```
l–,r++;
}

else
break;
}
}
if(curr_len==1)
cout<<"sorry no palindrome"<<endl;
else
cout<<max_len<<endl;
}
return(0);

}
```

Reply ↓ **0**

shivali
June 18, 2013 at 6:51 am

```
//edited://longest pa
#include
#include
#define lsfor(i,a,b) for(i=a;i<b;i++)

using namespace std;
char str[100];
int n,curr_len=0,max_len=1,l,r,t;

int main()
{
int i;
cout<<"enter no.of test cases:"<>t;
while(t–)
{
cout<<"please enter your string"<>str;
n=strlen(str);
if(n==1)
{cout<<"1"<<endl;
return(0);}
```

```
if(n==2)

{cout<<"2"<=0&&r<=n-1)
{
if(str[l]==str[r])
{
curr_len+=2;
if(max_len<curr_len)
max_len=curr_len;
l–,r++;
}

else
break;
}
}
if(max_len==1)
cout<<"sorry no palindrome"<<endl;
else
cout<<max_len<<endl;
}
return(0);

}
```

Reply ↓ **0**

jackyhou
July 3, 2013 at 1:05 am

```
int find_long_palindrome_line(char * str,char *substr)
{
if(str==NULL)
return -1;

int len=strlen(str);
if(len==0)
return 1;

int i=0;
int j=len-1;
int end = len-1;
```

```
int curindex=0;

int tmp_len=len*2+1;
char * tmp_allstr=(char *)malloc(sizeof(char)*(tmp_len));
int *i_arr=(int *)malloc(sizeof(int)*(tmp_len));

int index4_tmp_allstr=0;
for(int i=0;i<len;i++)
{
tmp_allstr[index4_tmp_allstr++]='#';
tmp_allstr[index4_tmp_allstr++]=str[i];
}
tmp_allstr[index4_tmp_allstr]='#';

memset(i_arr,0,sizeof(int)*(tmp_len));

for(int i=2;i0 && right <(tmp_len))
{
if(tmp_allstr[left]==tmp_allstr[right])
{
i_arr[i]++;
}
else
{
break;
}
}
else
{
break;
}
}
}

int findI=0;
int findMaxLen=0;
for(int i=2;ifindMaxLen)
{
findMaxLen=i_arr[i];
findI=i;
}

}
```

```
int realSubLen=0;
if(findMaxLen>0)
{
for(int i=findI-(findMaxLen)+1;i<=findI+(findMaxLen)-1;i=i+2)
{
substr[realSubLen++]=tmp_allstr[i];
}
substr[realSubLen]='';
}

free (tmp_allstr);
tmp_allstr=NULL;
free (i_arr);//=(int *)malloc(sizeof(int)*(tmp_len));
i_arr=NULL;
return 1;
}
```

Reply ↓ Report user **0**

JS
July 10, 2013 at 2:56 am

Won't the preprocess() take quadratic time? substr() is linear is time.

Reply ↓ **-1**

Saber
July 23, 2013 at 9:57 am

The algorithm you write is wrong. I believe it is because u type it faster than what u think:)
if P[i'] ≤ R – i,
then P[i] ← P[i']
else P[i] ≥ P[i']. (Which we have to expand past the right edge (R) to find P[i].

should be changed to:
if P[i'] < R – i,
then P[i] ← P[i']
else P[i] ≥ R – i. (Which we have to expand past the right edge (R) to find P[i].

Reply ↓ Report user **0**

shuaiyangyang
August 20, 2013 at 4:32 pm

if p[i] > R -i then p[i] should equal to R-i.

only if p[i] = R-i then you have to expand

Reply ↓ Report user 0

shuaiyangyang
August 20, 2013 at 4:30 pm

Hey 1337, The relation should be:

if P[i'] < R – i,

then P[i] ← P[i']

else if P[i] = R – i. (Which we have to expand past the right edge (R) to find P[i].

else p[i] = R – i

Reply ↓ Report user 0

Nam
January 5, 2015 at 2:51 pm

I agree with you.

However, for worse case scenario of strings such as “aaaaaaa”, the run time is O(n^2). I think this algorithm is O(n) on average.

Reply ↓ 0

Fentoyal
September 1, 2013 at 5:16 pm

This problem could be solved in O(n) time and O(1) space.
The test case attached here is what leetcode claims “wrong answer”, but I run it on my computer and it is exactly the same as the “expected answer”. I do not know why tho.
#include
#include

```
using namespace std;
class Solution {
int longestPalindromSubstrHelper(const string & str, bool is_even, int &cur_max_pivot, int & cur_max_radius)
{
int cur_radius = 0, cur_pivot = 0;
for (size_t i = 0; i < str.size(); ++i)
{

cur_radius = i – cur_pivot;
// cout<<cur_radius<<" "<<cur_pivot<<" "<<i<= 0 && str[cur_pivot – cur_radius + is_even] == str[i] &&cur_radius
>= cur_max_radius)
{
cur_max_radius = cur_radius;
cur_max_pivot = cur_pivot;
}
while ((cur_pivot – cur_radius + is_even < 0 ||str[cur_pivot – cur_radius +is_even] != str[i] ) && cur_pivot < i)
{
cur_pivot++;
cur_radius–;
//cout<<cur_radius<<" "<<cur_pivot<<" "<<i<<endl;
}
}
return 2 * cur_max_radius + !is_even;

}

public:
string longestPalindrome(string str) {
// Start typing your C/C++ solution below
// DO NOT write int main() function

int even_radius = 0, even_pivot = 0, odd_radius = 0, odd_pivot = 0;
int even_len = longestPalindromSubstrHelper(str, 1, even_pivot, even_radius );

int odd_len = longestPalindromSubstrHelper(str, 0, odd_pivot, odd_radius);
//cout<<even_len<<odd_len<= odd_len)
{
return str.substr(even_pivot – even_radius + 1, 2 * even_radius);
}

return str.substr(odd_pivot – odd_radius, 2 * odd_radius+1);

}
```

```
};

int main()
{
Solution s;
//cout<<longestPalindromSubstrHelper("ababab",1)<<endl;
cout<<s.longestPalindrome("321012321001232100123210012321001232100123210012321001232100123210
01232100123210012321001232100123210012321001232100123210012321001232100123210012321001232
10012321001232100123210012321001232100123210012321001232100123210012321001232100123210012
32100123210012321001232100123210012321001232100123210012321001232100123210012321001232100
12321001232100123210012321001232100123210012321001232100123210012321001232100123210012321
00123210012321001232100123210012321001232100123210012321001232100123210012321001232100123
21001232100123210012321001232100123210012321001232100123210012321001232100123210012321001
23210012321001232100123210012321001232100123210012321001232100123210012321001232100123210
01232100123210012321001232100123210012321001232100123210012321001232100123210012321001232
10012321001232100123210012321001232100123210012321001232100123210012321001232100123210012
32100123210012321001232100123210012321001232100123210012321001232100123210012321001232100
1232100123210123210012321001232100123210123")<<endl;
// cout<<longestPalindromSubstr("ababababa")<<endl;
}
```

Reply ↓ Report user **+1**

STANLEY
October 28, 2013 at 12:54 pm

if the string is atabcccbatabccccbata the cause the longest p. string's middle is 't' in central. But as your algorithm, the when the i =t R is larger than i; so the t is gonna to equals to the t at the second place. is 3. how could this figure out?

Reply ↓ **0**

stanley
October 28, 2013 at 1:07 pm

I'm not sure that cause you mean the R is the position which generated by the pivot's left bound of previous D. And when you met i < R it will equals to the min(), so If the string is atabcccbatabcccbata the when D at the position of the second c, the R is gonna to be the a behind the second t. But so when the i turn to the t, t should equals to the min(), but t is the pivot of the longest substring, so how this work in the algorithm? I've got a little confuse.

Reply ↓ Report user **0**

Pingback: [Leetcode]Longest Palindromic Substring | Wei's blog

Pingback: Manacher's algorithm (algorithm to find longest palindrome substring in linear time) | Ask Programming & Technology

Pingback: [LeetCode]Longest Palindromic Substring | MistySoul

Atul Kumar
November 15, 2013 at 1:28 am

Great !!!

Reply ↓ **0**

Mark
November 30, 2013 at 9:23 am

A misleading post.

Reply ↓ **+2**

Pingback: Longest Palindromic Substring | Jiting

Albert Chen
February 2, 2014 at 3:29 pm

Building suffix tree is O(n) and preprocessing a general tree for O(1) LCA queries is O(n).

We can build an extended suffix tree by inserting suffixes of reversed text into the same tree.

After these constructions, we enumerate the mid point in text (and we know the corresponding point in the reversed text). By looking at the LCA of the corresponding points in text and reversed text. We can decide in constant time that the longest palindrome from this mid point.

So we will be able to solve the problem in linear time (for odd length text.)

For more details, look into the book Algorithms on Strings, Trees and Sequences.

Reply ↓ **0**

Leet

February 3, 2014 at 4:27 am

```
for (int i = 1; i < t.length-1; i++) {

              while (t[i + (1 + p[i])] == t[i - (1 + p[i])])
                   p[i]++;
          }
```

this itself wil give the solution..
Whats the need of center and right part.. plz expalin me

Reply ↓ 0

Pingback: Solution to gamma2011 (Count-Palindromic-Slices) by codility | Code Says

Pingback: Longest Palindromic Substring | Elvis_Yu

online chesstle

February 15, 2014 at 10:26 am

s.substr((centerIndex – 1 – maxLen)/2, maxLen); is there a bug here as the rest of the code is treating P[i] as the length of the palindrome on either side of the center and not as the total length of the palindrome. maxlen is derived from P…

Reply ↓ 0

Acmerblog

March 22, 2014 at 2:54 am

great job.I'm not sure that cause you mean the R is the position which generated by the pivot's left bound of previous D. And when you met i < R it will equals to the min(), so If the string is atabcccbatabcccbata the when D at the position of the second c, the R is gonna to be the a behind the second t. But so when the i turn to the t, t should equals to the min(), but t is the pivot of the longest substring, so how this work in the algorithm? I've got a little confuse.

Reply ↓ +1

Donne

April 9, 2014 at 7:00 am

Why is the algoritmn O(n). The while loop where we attempt to expand the palindrome would need O(n) in worst

case. eg. in case of b of the center. We are indirectly traversing all the nodes in that loop. So wont the order be O(n^2) ?

Reply ↓ +1

Alex
April 11, 2014 at 1:36 am

Excellent post…. i don't find any other posts which is as good and elaborate as this !! Good job.

Reply ↓ 0

Pingback: (Leetcode) Longest Palindromic Substring | Changhaz's Codeplay

programmingmonkey
April 29, 2014 at 1:28 am

```
while (T[i + 1 + P[i]] == T[i - 1 - P[i]])
    P[i]++;
```

perhaps this part needs a little more index checking to avoid segment fault?

Reply ↓ Report user +1

Pingback: CodeNirvana: Palidromic Number | {.SMan.Soft.}

Pingback: CodeNirvana: Palidromic Numbers | {.SMan.Soft.}

Bob
June 3, 2014 at 6:37 pm

Thank you for your clear explanation！ ！

Reply ↓ 0

ash
August 4, 2014 at 1:32 am

for (int i = 1; i i) ? min(R-i, P[i_mirror]) : 0;

//There is a problem here for i=1 and first run(C=0) through the loop i_mirror = -1;
P[-1] … you are accessing wrong address..

Reply ↓ 0

killa
August 13, 2014 at 5:03 am

You've no idea how difficult it is for a chinese to understand your method. But you are brilliant. Thanks.

Reply ↓ 0

ccen
August 20, 2014 at 10:30 pm

learn a lot, thank you

Reply ↓ 0

Dinesh
August 30, 2014 at 7:01 am

Excellent O(N) time complexity solution. Thanks for the post 🙂

Reply ↓ +1

Pingback: 学习manacher（最长公共回文串算法）_58,AM,BBA,c#,CA,CTE,DC,EF,F1,FB,GE,HTML,Http,ie,IT,LeetCode,Logs,ROM,Set,Span,String,WP,XP,代码,位置,关系,博客,处理方法,字符串,学习,定义,怎么办,方法,时间,模式,比较,现在,矛盾

Pingback: LeetCode 最长回文子字符串 « 狐说的小站

Pingback: LeetCode in Swift: Longest Palindromic Substring – Guan Gui

B_Khan
November 2, 2014 at 4:40 am

Great Explanation. Thanks !

Reply ↓ 0

adiggo
November 6, 2014 at 12:40 pm

it is so brilliant, but so hard to come up with.

Reply ↓ Report user 0

Pingback: Data Structures and Algorithms Tutorials | TheShayna.Com

sagiv
November 20, 2014 at 11:35 pm

great post! thanks

Reply ↓ 0

Leo
November 28, 2014 at 11:30 am

Using symmetry to avoid recalculation. Simple and elegant.

Reply ↓ Report user 0

Pingback: [LeetCode] Longest Palindromic Substring 解题报告 | 水中的鱼(镜像)

Nick Nussbaum
January 5, 2015 at 5:59 pm

Rather than create S I made lambda to provide the equivalent of accessing S using the input string. A few more operations in the inner loop, as a tradeoff for space
I also added a little bounds checking

```
string LongestPalindrome(const string input)
{
    int c = 0;
    int max = 0;
    // create an indexed accessor for a virtual string S which has $a$b$c$ for  input st
    auto S_at = [&input](int index)->char { return ((index & 1) ? input[index / 2] : '$'
    int sizeP = (input.length() * 2) + 1;
    int* P = new int[sizeP];
    P[0] = 0;
```

```
    // find longest Palindromes forcentered on each index in S
    for (int i = 1; i  i) ? min(P[c - (i - c)], max - i) : 0;
        // Try to expand Palindrome but not past string boundaries
        int bounds = min(sizeP - i - 1, i - 1);
        while (bounds-- >= 0 && S_at(i + P[i] + 1) == S_at(i - P[i] - 1))
        {
            P[i]++;
        }
        // If palindrome was extend past max then update Center to i and update the righ
        if (i + P[i] > max)
        {
            c = i;
            max = i + P[i];
        }
    }
    auto maxP = std::max_element(P, P + sizeP);
    int start = (maxP - P - *maxP)/2;
    return input.substr(start, *maxP);
}
```

Reply ↓ 0

Nick Nussbaum
January 5, 2015 at 6:04 pm

The seems to have mangled the code badly

```
 string LongestPalindrome(const string input)
{
int c = 0;
int max = 0;
// create an indexed accessor for a virtual string S which has $a$b$c$ for input
string abc
auto S_at = [&input](int index)->char { return ((index & 1) ? input[index / 2] :
'$'); };
int sizeP = (input.length() * 2) + 1;
int* P = new int[sizeP];
P[0] = 0;
// find longest Palindromes for centered on each index in S
for (int i = 1; i i) ? min(P[c - (i - c)], max - i) : 0;
// Try to expand Palindrome but not past string boundaries
int bounds = min(sizeP - i - 1, i - 1);
while (bounds-- >= 0 && S_at(i + P[i] + 1) == S_at(i - P[i] - 1))
```

```
{
P[i]++;
}
// If palindrome was extend past max then update Center to i and update the
right edge
if (i + P[i] > max)
{
c = i;
max = i + P[i];
}
}
auto maxP = std::max_element(P, P + sizeP);
int start = (maxP - P - *maxP)/2;
return input.substr(start, *maxP);
}
```

Reply ↓ 0

Nick Nussbaum
January 5, 2015 at 6:11 pm

Still mangling things.
Here's the mangled lines of the first post

```
    // find longest Palindromes centered on each index in S

    for (int i = 1; i  i) ? min(P[c - (i - c)], max - i) : 0;
```

Reply ↓ 0

RFonseca
July 6, 2015 at 12:22 am

Hi Nick

I'm eager to try out your implementation but i still dont see the un-mangled code.

Best

Reply ↓ 0

Pingback: Python：最长回文子串

Mazeryt
February 13, 2015 at 1:01 pm

Hello, I have question about proof about time complexity of this solution?
Because You had a embedded while loop that seems that can for each element with index i, run i-times to verify the equality. What can be the worst case of this loop? and accurate time complexity?

I will be grateful for more details

Reply ↓ 0

Pingback: Longest Palindromic Substring | moonstonelin

Md Mishfaq Ahmed
February 17, 2015 at 2:35 pm

Nice algorithm

Reply ↓ Report user 0

Pingback: The Manacher's Algorithm for finding longest palindrome substring | codelearn

Pingback: LeetCode(5) Longest Palindromic Substring – 剑客|关注科技互联网

jerry
May 19, 2015 at 6:22 am

Greet write-up. Thanks a lot. And the trick shown in your code is very awesome!

Reply ↓ 0

Pingback: Longest Palindrome Substring. – The Programming Club @ DA-IICT

SHUBHAM GUPTA
June 14, 2015 at 5:23 am

Your code is giving the output for "abcabcacba" as "abcacba" (length=7)

However it should be “abcaacba” (length=7)

please explain

Reply ↓ 0

SHUBHAM GUPTA
June 14, 2015 at 5:23 am

Correction
However it should be “abcaacba” (length=8)

Reply ↓ 0

Yash
July 9, 2015 at 1:35 pm

Excellent Post. Will love to read your other posts 🙂

Reply ↓ -1

Pingback: LeetCode[148]: Longest Palindromic Substring | 形&影

Mahmoud Emam
August 1, 2015 at 7:56 am

this code is an improvement over the mentioned algorithm in terms of memory usage !

```
string S;
     char get(int i)
     {
         return i & 1 ? S[i >> 1] : '#';
     }
     string longestPalindrome(string s) {
         S = s;
         int len = (s.size() << 1)+ 1;
         vector p(len, 0);
         int c = 0; int r = 0;
         int maxi = 0, maxLen = 0;
         for (int i = 1; ii) ? min(r - i, p[i_mirror]) : 0;

             while ((i - p[i] - 1) >= 0 && (i + p[i] + 1)r)
             {
                 r = i + p[i];
                 c = i;
             }
```

```
            if (p[i] > maxLen)
                maxLen = p[i],maxi = i;

        }

        return s.substr(abs(maxLen-maxi) / 2, maxLen);
    }
```

Reply ↓ 0

lestrois
August 10, 2015 at 4:03 am

Should this statement “In this case, since P[21] ≠ P[1], we conclude that P[i] = 5.” be “In this case, since T[21] ≠ T[1], we conclude that P[i] = 5.”? Thanks.

Reply ↓ 0

Pingback: Manacher Algorithm-Longest Palindrome Substring | Xinli Blog

Rick Mac Gillis
October 1, 2015 at 3:33 pm

Thanks for your amazing article! I cleaned your code in a port to the Hack language. Hopefully the cleaner version will help other developers to understand how the algorithm works at a code level. Feel free to link to it in your article.

Check it out: https://github.com/cozylife/hackfastalgos/blob/master/lib/palindrome.php

Reply ↓ 0

Kool
October 18, 2015 at 11:37 pm

Sorry, I still don’t see why linear, although I believe it is. 🙂

One center change doesn’t count for the number of comparisons. The best upper bound I can see is O(M*N) where M is the max length of palindrome and N is the number of elements. Any more insight?

Reply ↓ 0

Kool

October 18, 2015 at 11:42 pm

Got it after re-read the amortized r

Reply ↓ 0

Russell

October 19, 2015 at 2:51 pm

The while loop condition should be as bellow.
while (i-1-p[i] >= 0 && i + 1 + p[i] < n && T[i + 1 + p[i]] == T[i-1-p[i]])

Reply ↓ 0

Kool

October 24, 2015 at 10:41 am

The major undesired part of this algorithm is the need to identify a distinguishable char to mock the original string. Another algorithm, though complicated, is to extend the right endpoint instead of the center at each iteration, published in 1990., with no need to mock the original string.

Reply ↓ 0

Kool

October 24, 2015 at 10:43 am

The algorithm is published at the book Beauty is our Business

https://books.google.com/books?id=20ThBwAAQBAJ&pg=PA410&lpg=PA410&dq=udding+palindrome&source=bl&ots=tpOXm_855_&sig=Mrv3NG1ODEUiH50Uv9cIDknNBh4&hl=en&sa=X&ved=0CCgQ6AEwAmoVChMI6Yuup9LbyAIVE8xjCh1KzQE7#v=onepage&q=udding%20palindrome&f=false

Reply ↓ 0

John

December 8, 2015 at 9:45 am

Hello,
I am trying to get the three longest palindromes in a string. I was expecting that the array P would give all the palindromes found? But for the input baab the array produced is [0, 1, 0, 1, 4, 1, 0, 1, 0].

Expected palindromes: {baab,aa,b,a}
Why is the algorthm not considering a palindrome with length 2 (aa)?

Reply ↓ 0

Pingback: 005: Longest Palindromic Substring – Hello World

Qianqian
January 26, 2016 at 2:27 pm

Sorry, but I am confused by the summary, it says:

if P[i'] ≤ R – i,
then P[i] ← P[i']
else P[i] ≥ P[i']. (Which we have to expand past the right edge (R) to find P[i].

But the example above the summary has P[i'] = 7 while P[i] = 5 for i = 15. So I think you meant to write P[i] >= R-i, which also makes sense to me…

Reply ↓ 0

Pingback: 5. Longest Palindromic Substring – lofsöngur

Kate
May 12, 2016 at 8:35 pm

to perform these same procedures but the program R as serious ?

Reply ↓ 0

songlili
May 17, 2016 at 5:04 am

To embed your code, please use

```
your code here
```

.

Reply ↓ 0

Pingback: LeetCode OJ (C#) – Longest Palindromic Substring | miafish

Pingback: 5. Longest Palindromic Substring – Pengfei Ren

Daozhuang
July 26, 2016 at 4:00 am

In this sentence:

else P[i] ≥ P[i']. (Which we have to expand past the right edge (R) to find P[i].

Should P[i'] be (R-i)?

Reply ↓ 0

Amirtha
January 5, 2017 at 4:02 pm

Hi ,

Thanks for explaining the algorithm clearly!
I have a question.
Using this Algorithm , we will be able to calculate the Longest Common Substring definitely but with the method that returns the longest palindromic Substring, will we be able to list all the Longest Substring ?

For Example : String input = abaaa
Longest Substring length = 3
Longest Substring 1 = aba
Longest Substring 2 = aaa

In this case should we should be storing all the center values and later return all possible substrings.

Thanks,
Amirtha

Reply ↓ 0

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

To embed your code, please use <code>your code here</code>.

You may use the <code> tag to embed your code.

Name *

Email *

Website

Post Comment