Longest Palindromic Substring Part I

November 20, 2011 by 1337c0d3r 96 Replies

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

This interesting problem has been featured in the famous Greplin programming challenge, and is asked quite often in the interviews. Why? Because this problem can be attacked in so many ways. There are five different solutions that I am aware of. Are you up to the challenge?

Head over to **Online Judge** to solve it now! (you may submit either C++ or Java solution)

Hint:
First, make sure you understand what a palindrome means. A palindrome is a string which reads the same in both directions. For example, “aba” is a palindome, “abc” is not.

A common mistake:
Some people will be tempted to come up with a quick solution, which is unfortunately flawed (however can be corrected easily):

> Reverse S and become S’. Find the longest common substring between S and S’, which must also be the longest palindromic substring.

This seemed to work, let’s see some examples below.

For example,
S = “caba”, S’ = “abac”.
The longest common substring between S and S’ is “aba”, which is the answer.

Let’s try another example:

S = “abacdfgdcaba”, S’ = “abacdgfdcaba”.
The longest common substring between S and S’ is “abacd”. Clearly, this is not a valid palindrome.

We could see that the longest common substring method fails when there exists *a reversed copy of a non-palindromic substring in some other part of S*. To rectify this, each time we find a longest common substring candidate, we check if the substring’s indices are the same as the reversed substring’s original indices. If it is, then we attempt to update the longest palindrome found so far; if not, we skip this and find the next candidate.

This gives us a O(N^2) DP solution which uses O(N^2) space (could be improved to use O(N) space). Please read more about Longest Common Substring here.

Brute force solution, O(N^3):
The obvious brute force solution is to pick all possible starting and ending positions for a substring, and verify if it is a palindrome. There are a total of C(N, 2) such substrings (excluding the trivial solution where a character itself is a palindrome).

Since verifying each substring takes O(N) time, the run time complexity is O(N^3).

Dynamic programming solution, O(N^2) time and O(N^2) space:
To improve over the brute force solution from a DP approach, first think how we can avoid unnecessary re-computation in validating palindromes. Consider the case “ababa”. If we already knew that “bab” is a palindrome, it is obvious that “ababa” must be a palindrome since the two left and right end letters are the same.

Stated more formally below:

> Define P[i, j] ← true **iff** the substring S_i … S_j is a palindrome, otherwise false.

Therefore,

> P[i, j] ← (P[i+1, j-1] **and** $S_i = S_j$)

The base cases are:

> P[i, i] ← true
> P[i, i+1] ← ($S_i = S_{i+1}$)

This yields a straight forward DP solution, which we first initialize the one and two letters palindromes, and work our way up finding all three letters palindromes, and so on…

This gives us a run time complexity of O(N^2) and uses O(N^2) space to store the table.

```
string longestPalindromeDP(string s) {
  int n = s.length();
  int longestBegin = 0;
  int maxLen = 1;
  bool table[1000][1000] = {false};
  for (int i = 0; i < n; i++) {
    table[i][i] = true;
  }
  for (int i = 0; i < n-1; i++) {
    if (s[i] == s[i+1]) {
      table[i][i+1] = true;
      longestBegin = i;
      maxLen = 2;
    }
  }
  for (int len = 3; len <= n; len++) {
    for (int i = 0; i < n-len+1; i++) {
      int j = i+len-1;
      if (s[i] == s[j] && table[i+1][j-1]) {
        table[i][j] = true;
        longestBegin = i;
        maxLen = len;
      }
    }
  }
  return s.substr(longestBegin, maxLen);
}
```

Additional exercise:

Could you improve the above space complexity further and how?

A simpler approach, O(N^2) time and O(1) space:

In fact, we could solve it in O(N^2) time without any extra space.

We observe that a palindrome mirrors around its center. Therefore, a palindrome can be expanded from its center, and there are only 2N-1 such centers.

You might be asking why there are 2N-1 but not N centers? The reason is the center of a palindrome can be in between two letters. Such palindromes have even number of letters (such as “abba”) and its center are between the two ‘b’s.

Since expanding a palindrome around its center could take O(N) time, the overall complexity is O(N^2).

```
string expandAroundCenter(string s, int c1, int c2) {
  int l = c1, r = c2;
  int n = s.length();
  while (l >= 0 && r <= n-1 && s[l] == s[r]) {
    l--;
    r++;
  }
  return s.substr(l+1, r-l-1);
}
```

```
string longestPalindromeSimple(string s) {
  int n = s.length();
  if (n == 0) return "";
  string longest = s.substr(0, 1);  // a single char itself is a palindrome
  for (int i = 0; i < n-1; i++) {
    string p1 = expandAroundCenter(s, i, i);
    if (p1.length() > longest.length())
      longest = p1;

    string p2 = expandAroundCenter(s, i, i+1);
    if (p2.length() > longest.length())
      longest = p2;
  }
  return longest;
}
```

Further Thoughts:

Does an O(N) solution exist? You bet! However, it is not trivial and requires some very clever observation. The O(N) solution is explained in my next post.

» **Continue reading Longest Palindromic Substring Part II.**

Rating: 4.9/**5** (164 votes cast)

Longest Palindromic Substring Part I, 4.9 out of 5 based on 164 ratings

← Regular Expression Matching

Longest Palindromic Substring Part II →

96 thoughts on “Longest Palindromic Substring Part I”

Programming Praxis
November 11, 2011 at 2:09 pm

I solved this problem at my blog.

Reply ↓ +3

1337c0d3r Post author
November 12, 2011 at 1:23 am

Great job coding up the O(n) solution!

+2

Reply ↓

Daniel Sobral
November 11, 2011 at 8:50 pm

I solved it on codereview. I tried for a functional version with better than quadratic performance, since the one-liners were all quadratic.

http://codereview.stackexchange.com/questions/5241/more-functional-way-of-writing-this-palindrome-extractor/5260#5260

I'm afraid mine was definitely not one-liner. 🙂 The first version was half the size of that, but code density was way too high.

Reply ↓ **-2**

jcleetcode
November 12, 2011 at 7:18 am

I also write a O(n) algorithm as Programming Praxis metioned in his URL. Submit to online judge system.

Reply ↓ Report user **0**

vaibhav
November 12, 2011 at 4:34 pm

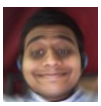
What about reversing the original string and then finding the longest common substring of both?

Reply ↓ **-28**

Peter Liu
October 2, 2013 at 9:50 pm

have you ever looked at the post..

Reply ↓ Report user **+21**



Praveen
November 12, 2011 at 5:50 pm

@vaibhav that would not work. The string "abaaacedfaaabaxyz" would output "abaa" which is not the desired result.

Reply ↓ 0

durbin

November 16, 2011 at 12:24 am

```
String longestPalindrome(String s) {
      int len = s.length();
      if (len == 0) {
          return null;
      }

      int maxPalindromeLen = 0;
      int centerPos = 0;

      // check palindrome centered at s[i]
      for (int i = 0; 2*(len - i) - 1 > maxPalindromeLen; i++) {
          int tmpLen = 0;
          int j = 0;

          while (j <=i && (i + j)  maxPalindromeLen) {
              maxPalindromeLen = tmpLen;
              centerPos = i;
          }
          }
      }

      // check palindrome centered at s[i] and s[i+1]
      for (int i = 0; 2*(len - i) -2 > maxPalindromeLen; i++) {
          int j = 0;
          int tmpLen = 0;
          while (j <=i && (i + j + 1)  maxPalindromeLen) {
              maxPalindromeLen = tmpLen;
              centerPos = i;
          }
          }

      }

      return s.substring(centerPos-(maxPalindromeLen -1)/2, centerPos+maxPalindromeLen/2 + 1);
}
```

Reply ↓ -3

wayne

November 16, 2011 at 5:07 pm

I think that mine is linear.

```
import java.util.*;
public class palindromeSolution{
    public static String input="adfasdfcabbacdfadsfasdfa";
    public static Map statistic = new HashMap();
    public static Map allPalinDromes = new HashMap();
```

```
    public static void main(String[] args){
        prepareInfo();
        //System.out.println("shang:statistic:"+statistic);
        pickPalinDromes();
    }
    public static void prepareInfo(){
        for(int i=0;i&lt;input.length();i++){
            char a = input.charAt(i);
            //System.out.println(&quot;shang:char:&quot;+a);
            List positions = (List) statistic.get(Character.toString(a));
            if(positions == null){
                positions = new ArrayList();
            }
            positions.add(i);
            statistic.put(Character.toString(a),positions);

        }
    }
    public static void pickPalinDromes(){
        for(Object entry:  statistic.entrySet()){
           String endChar = (String) ((Map.Entry)entry).getKey();
           List positions = (List) ((Map.Entry)entry).getValue();
           if(positions.size()&lt;=1){
               continue;
           }
           loopThroughPositions(positions);
        }
    }
    public static void loopThroughPositions(List positions){
            for(int i=0;i&lt;positions.size();i++){
               if(i==positions.size()){
                   continue;
               }
               Integer start = (Integer) positions.get(i);
               for(int ii=i+1;ii2){
                       continue;
                   }
                   findMaxPalinDrome(start,endd);


               }

            }
    }
    public static void findMaxPalinDrome(Integer start, Integer endd){
        boolean stopFlag = true;
        while(stopFlag){
           start--;
           endd++;
           if(start&lt;0){
               continue;
           }
           char startChar = input.charAt(start);
           char endChar = input.charAt(endd);
           if(startChar != endChar){
               stopFlag=false;
               allPalinDromes.put(start+1,endd-1);
               System.out.println(&quot;yes, we found one:&quot;+input.substring(start+1,endd
           }
```

```
        }
    }
}
```

Reply ↓ +2

rachel

November 17, 2011 at 4:49 am

Java version:

```
public String longestPalindrome(String s)
    {
        char[] c = s.toCharArray();
        int maxLen = 0, len=0;
        int begin = 0;
        for(int k=0;k=0 &amp;&amp; k+len+1 maxLen)
            {
                begin = k-(len-1)-1;
                maxLen = len*2+1;
            }
            len = 0;
            while(k&gt;=len &amp;&amp; k+len+1 maxLen)
            {
                begin = k-(len-1);
                maxLen = len*2;
            }
        }
        return new String(c,begin,maxLen);
    }
```

Reply ↓ +1

rachel

November 17, 2011 at 6:17 am

```
public String longestPalindrome(String s)
{
        char[] c = s.toCharArray();
        int maxLen = 0, len=0;
        int begin = 0;
        for(int k=0;k<s.length()-1;k++)
        {
            len = 0;
            while(k-len-1>=0 && k+len+1<s.length() && c[k-len-1]==c[k+len+1])
            {
                len+=1;
            }
            if(len*2+1 > maxLen)
            {
                begin = k-(len-1)-1;
                maxLen = len*2+1;
```

```
            }
            len = 0;
            while(k>=len && k+len+1<s.length() && c[k-len]==c[k+len+1])
            {
                len+=1;
            }
            if(len*2 > maxLen)
            {
                begin = k-(len-1);
                maxLen = len*2;
            }
        }
        return new String(c,begin,maxLen);
}
```

Reply ↓ +1

1337c0d3r Post author
November 17, 2011 at 4:56 am

@durbin @wayne @rachel I just added syntax highlighting for your code. This is an experimental feature. Just FYI, you can syntax highlight using [lang]your code here…[/lang]. Replace lang with your favorite language, in this case, lang=java.

Reply ↓ 0

quasar
December 6, 2011 at 10:30 am

Is a single char a palindrome? And I think there is a bug in your second solution. When there is a single char string input, the expandAroundCenter() will return s.substr(1, 0). Is that correct?

Reply ↓ +1

1337c0d3r Post author
December 6, 2011 at 10:58 am

Very good observation. I missed that completely.

If you notice carefully however, the code still runs fine. In C++, for a single char string input, s.substr(1,0) returns an empty string. In general, for s with length of n, s.substr(n, k) will always return an empty string for k >= 0. http://www.cplusplus.com/reference/string/string/substr/

However, my code is not written in a good way. Good code should reflect clearly its intentions. Even though it does work, it is highly misleading as it seem like it is out of bounds. Thanks, and I have refined the code.

+1

Reply ↓

garima
April 13, 2012 at 6:26 pm

Pls correct me if I am wrong here..

If we enter single character, we are not going to enter in the for loop itself as length is 1 then and the loop condition
i=0; i<n-1; i++
restricts to enter.
Then, how come expandAroundCenter() function is called?

Reply ↓ **0**

kevin2
September 22, 2013 at 7:55 pm

good good code

-2

kevin3
September 22, 2013 at 7:56 pm

bad code

-2

kevin
September 22, 2013 at 7:54 pm

Good code

Reply ↓ **-2**

Prongs
November 18, 2011 at 9:48 am

In JAVA, O(n*n):

public class Palindrome {

```
public static void main(String[] args) throws IOException {
int min=-1;
String s,sub,rev;
StringBuffer temp;
ArrayList arr = new ArrayList();
int len;

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter a string");
s = br.readLine();

if(s.length() == 0)
System.out.println("None");

len = s.length();

for(int i = 0; i < len; i++)
{
for(int j = i; j <= len; j++)
{
sub = s.substring(i,j);
if(sub.length() min){
min=sub.length();
arr.add(sub);
}
}
}
}
System.out.println("Substring with longest palindrome – "+arr.get((arr.size()-1)));
}
}
```

Reply ↓ **0**

Pingback: Longest Palindromic Substring Part II | LeetCode

Qi Li
December 21, 2011 at 6:02 pm

```
string longestPalindrome(string s) {
// Start typing your C/C++ solution below
// DO NOT write int main() function
// NO #include's are required
if (s.size() == 1) {
```

```
return s;
}
string t = “”;
for (int i = 1; i = 0 && (i + j + 1) t.size()) {
t = r;
}
j = 0;
while (s[i – 1 – j] == s[i + j] &&
(i – 1 – j) >= 0 && (i + j) t.size()) {
t = r;
}
}
return t;
}
```

Reply ↓ **0**

Pingback: Palindrome Number I LeetCode

Razor
February 17, 2012 at 3:30 pm

I think there is a bug in your 2nd code again, Say when s = “abba” and i = 1 . I come to p2 expandString takes(s,1,2) and in the while loop l becomes -1 and r becomes 4
So when returning the substring u return s.substr(-1+1,4-(-1+1)) = s.substr(0,4) which is out of bounds.

Reply ↓ **-2**

1337c0d3r Post author
February 17, 2012 at 5:33 pm

s.substr(0,4) returns “abba”, which means start at index 0 and its length is 4.

Reply ↓ **0**

Razor
February 17, 2012 at 3:32 pm

it should be while(l > 0 && r < n-1 && s[l] == s[r])

Reply ↓ **0**

1337c0d3r Post author
February 17, 2012 at 5:34 pm

If you do this you would not compare the two characters at the end for equality. This would miss out some palindromes.

Reply ↓ 0

Razor
February 17, 2012 at 3:42 pm

and also return s.substr(l,r-l-1)

Reply ↓ 0

Pingback: longest palindrom | laibinshi

Pingback: Common palidromes | Tamihughson

Neevan
March 16, 2012 at 10:18 pm

Thnx for clear explanation….

Reply ↓ 0

Neevan
March 16, 2012 at 10:18 pm

Thnx for succinct explanation….

Reply ↓ 0

Naveen
March 16, 2012 at 10:57 pm

Hey 1337c0d3r,
Being a non-native English speaker i find one para little confusing ..could you please rephrase the below paragraph in simple English ..Thank you in advance…

” We could see that the longest common substring method fails when there exists this …… ” till the end of that paragraph ..particularly what do you mean by this “a reversed copy of a non-palindromic substring in some other part

of S”…

Reply ↓ 0

Shundan Xiao
May 4, 2012 at 7:52 pm

Hi, I think there is a bug in this method:

string expandAroundCenter(string s, int c1, int c2) {

// that’s what to added to make it correct. otherwise if you have
//s=”bc”, c1=0 and c2=1, you won’t be able to get into the while
//loop, but when you return, you are returning based on the
//modified index.

if(s[c1]!=s[c2]){
return “”;
}
int l = c1, r = c2;
int n = s.length();
while (l >= 0 && r <= n-1 && s[l] == s[r]) {
l–;
r++;
}
return s.substr(l+1, r-l-1);
}

Reply ↓ Report user 0

Omar Mohammad Othman
May 7, 2012 at 5:59 am

Could you please explain this paragraph again? I really didn’t get it…

To rectify this, each time we find a longest common substring candidate, we check if the substring’s indices are the same as the reversed substring’s original indices. If it is, then we attempt to update the longest palindrome found so far; if not, we skip this and find the next candidate.

Reply ↓ 0

Rohit
May 7, 2012 at 9:46 pm

The DP method fails for your counterexample in the beginning of the article. Try “abartaba”

Reply ↓ 0

reader

May 26, 2012 at 11:45 am

I can not understand the table[1000][1000]

Why 1000? How did you come up with 1000?Could you please explain?

Reply ↓ 0

Sherry

June 21, 2012 at 11:46 am

I have a solution and passed all the test cases, I used two for loops and a recursive function inside the second loop, does this mean this is O(N3) in time??But my space is O(1), I think. Here is my code:

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

        int firstIndex=0,lastIndex=0,nextFIndex = 0,nextLIndex=0;

        if(s.length() == 1)
            return s;
        else
        {
            for(int i=0; i < s.length()-1; i++)
            {
                nextFIndex = i;
                nextLIndex = 0;
                for(int j=s.length()-1; j>i; j--)
                {
                    if(IsPalindrome(s, i, j))
                    {
                        nextLIndex = j;
                        break;
                    }
                }

                if(nextLIndex!=0 && (nextLIndex-nextFIndex+1 > lastIndex-firstIndex+1))
                {
                    firstIndex = nextFIndex;
                    lastIndex = nextLIndex;
                }
            }
```

```
            if(lastIndex != 0)
                return s.substr(firstIndex, lastIndex-firstIndex+1);
            else
                return "";
        }

    }

private:
    bool IsPalindrome(string& s, int firstIndex, int lastIndex)
    {
        if(s[firstIndex] == s[lastIndex])
        {
            if(firstIndex == lastIndex || (firstIndex+1)==lastIndex)
                return true;
            else
                return IsPalindrome(s, firstIndex+1, lastIndex-1);
        }
        else
            return false;
    }

};
```

Reply ↓ Report user 0

grubbyfan
September 15, 2012 at 12:24 pm

java version, passed both small and large,
running time is n^2 , O(1) space

public String longestPalindrome(String s) {
int frontIndex = 0;
int backIndex = 0;
for (int i = 1; i = 0 && front frontIndex – backIndex + 1) {
frontIndex = front – 1;
backIndex = back + 1;
}
}
back = i – 1; front = i;
if (front + 1 = 0 && front + 1 frontIndex – backIndex + 1) {
backIndex = back + 1;
frontIndex = front;
}
}
}
String str = s.substring(backIndex, frontIndex + 1);
return str;

}

Reply ↓ Report user 0

grubbyfan
September 15, 2012 at 12:25 pm

public String longestPalindrome(String s) {
int frontIndex = 0;
int backIndex = 0;
for (int i = 1; i = 0 && front frontIndex – backIndex + 1) {
frontIndex = front – 1;
backIndex = back + 1;
}
}
back = i – 1; front = i;
if (front + 1 = 0 && front + 1 frontIndex – backIndex + 1) {
backIndex = back + 1;
frontIndex = front;
}
}
}
String str = s.substring(backIndex, frontIndex + 1);
return str;
}

Reply ↓ Report user 0

Pingback: Longest Palindromic Subsequence I This is how Cheng grows up

Ashwani Kumar
November 5, 2012 at 2:23 am

Hello:

I am not able to get the following lines
“You might be asking why there are 2N-1 but not N centers? The reason is the center of a palindrome can be in between two letters.” Here, I am not clear why there are 2N-1 centers. Can somebody explain with some example?

Reply ↓ +1

Kevin Hsu

November 4, 2014 at 12:36 pm

I think the idea is that we don't know if the palindrome we are looking for is even or odd in length. Consider the following 2 examples:

s = "aba"
In this case, we need to call expandAroundCenter(s, i, i) to be able to successfully find this palindrome.

s = "abba"
In this case however, if we use the same approach as above, we would examine "abb", "bba" instead of "abba".
Thus, we need to call expandAroundCenter(s, i, i+1).

In summary, since we don't know if the length we are looking for is even or odd, we will check for both, and return the longest substring.

Reply ↓ **+1**

Aks
January 3, 2013 at 10:33 pm

The total number of brute force strings are not nc2. You have mentioned in the brute force solution as total number of substrings are nc2 which is incorrect.

Reply ↓ **0**

Aks
January 3, 2013 at 10:35 pm

I think they are

nc1 + Sum (i = 1 to n-1) (nci) / 2 + ncn

where nci = n combination i

Reply ↓ **0**

Mony Sim
January 20, 2013 at 11:56 am

This is the shortest solution.
bool isPalindrome(string s){

```
int i=0;
int len=s.length();
while(i<len/2)
{
if(s[i]!=s[len-i-1)
return false;
}
return true;
}
```

Reply ↓ Report user **0**

Mony Sim
January 20, 2013 at 12:27 pm

This is the shortest solution.

```
bool isPalindrome(string s){

int i=0;
int len=s.length();
while(i<len/2)
{
if(s[i]!=s[len-i-1)
return false;
i++;
}
return true;
}
```

Reply ↓ Report user **+2**

Kahn
February 25, 2013 at 7:00 pm

you sure get the title correctly?

Reply ↓ Report user **+1**

Yitong Zhou
June 4, 2013 at 5:51 pm

I think I can come up with a O(nlogn) solution, by binary searching the longest length and use Hashtable to check palindromic string mactching. Really curious to know how the O(n) works. Because as far as I know in wiki, the suffix

tree method actually takes O(nlog^2n), because although traversing suffix tree is O(n) time, building suffix tree needs O(nlog^2 n).

Reply ↓ Report user 0

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

Pingback: Longest Palindromic Substring Part I cloris1000

jackyhou
July 2, 2013 at 11:19 pm

```
int find_long_palindrome(char * str,int *maxlen,int * begin4max)
{
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;

printf("curindex=%d,end=%d\r\n",curindex,end);
while( (end-curindex)>*maxlen )
{
i=curindex;
j=end;
printf("curindex=%d\r\n",curindex);
while(1)
{
i=curindex;
int beginj=j;
while(icurindex && i>=j )
{
*begin4max=curindex;
*maxlen=beginj-curindex+1;
curindex++;
break;
}
```

```
j–;

if(i>=j)
{
curindex++;
break;
}
}
}
return 1;
}
```

Reply ↓ Report user 0

jackyhou
July 2, 2013 at 11:48 pm

it should be like this:

```
int find_long_palindrome(char * str,int *maxlen,int * begin4max)
{
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;

printf("curindex=%d,end=%d\r\n",curindex,end);
while( (end-curindex)>*maxlen )
{
i=curindex;
j=end;
printf("curindex=%d\r\n",curindex);
int beginj=j;
while(1)
{
i=curindex;
j=beginj;
while(icurindex && i>=j )
```

```
{
int tmp_begin4max=curindex;
int tmp_maxlen=beginj-curindex+1;

if(tmp_maxlen>(*maxlen))
{
*begin4max=curindex;
*maxlen=beginj-curindex+1;

char *pstr13=(char *)malloc(sizeof(char) * (*maxlen+2));
int realwritelen=_snprintf(pstr13,(*maxlen)+1,"%s",str+(*begin4max));
if(realwritelen>0)
{
pstr13[realwritelen]='';
}

printf("\r\n–writestris=[%s]——maxlen=[%d],begin4max=[%d]-realwritelen=[%d]-
\r\n",pstr13,*maxlen,*begin4max,realwritelen);
free(pstr13);
pstr13=NULL;
}

curindex++;
break;
}

//j=beginj-1;
beginj–;

if(i>=j)
{
curindex++;
break;
}
}
}
return 1;
}
```

Reply ↓ Report user **0**

marutikutre
July 15, 2013 at 11:26 am

```
public class LongestPallindrome {
public static void main(String[] args) {
String str = “abac”;
String longestPal = “”;
for (int i = 0; i longestPal.length())
{
longestPal = palinString;
}
}
System.out.println(“Finally **” + longestPal);

}

static String isPalindome(String str, int mid) {
boolean flag = true;
boolean isStart = true;
char[] string = str.toCharArray();
int length = string.length – 1;
int left = mid, right = mid;
while (flag && left > 0 && right < length)
{
if (isStart && string[left] == string[right + 1])
{
left–;
right = right + 2;
} else if (isStart && string[left – 1] == string[right])
{
left = left – 2;
right++;
} else if (string[left] == string[right])
{
left–;
right++;
} else
{
flag = false;
}
isStart = false;
}
if (flag && (left < 0 || right == length))
{
return str.substring(left, right + 1);
}
return str.substring(left , right+1);
```

```
}
}
```

Reply ↓ 0

Dan
July 15, 2013 at 11:49 am

“To rectify this, each time we find a longest common substring candidate, we check if the substring’s indices are the same as the reversed substring’s original indices. If it is, then we attempt to update the longest palindrome found so far; if not, we skip this and find the next candidate.”

This is wrong – for ABAXYZXABA – the longest common substring candidate will be ABAX and it will have expected indices – yet it’s not a palindrome. I don’t see how this can be resolved by reversing the string and looking for longest common substring.

Reply ↓ 0

AAS
August 13, 2013 at 12:09 pm

“To rectify this, each time we find a longest common substring candidate, we check if the substring’s indices are the same as the reversed substring’s original indices. If it is, then we attempt to update the longest palindrome found so far; if not, we skip this and find the next candidate.”

Can you please explain this through code?

Reply ↓ 0

GT
September 3, 2013 at 3:49 pm

Even I didnt understand the argument given in the post w.r.t using LCSubstring(string, reverse(string). But one way I see that we could make it work is if we check if the substring is a palindrome or not. This essentially makes it a n^3 solution, however. I would think that the idea of checking indices upon using LCS(string, reverse(string)) would also turn out to be O(n^3). Any clarification on this regard would be appreciated. [As it may improve the clarity in the original post]

But otherwise, the other solution looks clearly explained to me.

Reply ↓ 0

Pingback: leetcode总结无止境系列之动态规划及比较 – Crystal

Ramgopal Tanikella
August 21, 2013 at 7:18 pm

The last line in the longestPalindromeDP method should be:
return s.substr(longestBegin, longestBegin + maxLen);

Reply ↓ 0

Pingback: Finding longest palindrome in a string « ..MindWrite..

GP
November 1, 2013 at 1:28 pm

Great post!

I think you can add a check for empty string in the expandAroundCenter.

Thanks!

Reply ↓ 0

Li
November 23, 2013 at 8:37 pm

thanks this has been helpful

Reply ↓ Report user 0

vismaster
December 9, 2013 at 2:40 pm

Here is my O(N2) code:

#include

class Solution {
public:
string longestPalindrome(string s) {
if (s.length() <= 1) return s;
if (s.length() <= 2 && s[0] == s[1]) return s;

```
int start, length;
int maxStart = 0;
int maxLength = 0;

for (int idx=0; idx<s.length(); idx++) {
int expandable = min(idx, (int)s.length()-idx-1);
// Center at idx;
for (int expand=1; expand maxLength) {
maxStart = start;
maxLength = length;
}
// Center between idx and idx + 1;
expandable = min(idx, (int)s.length()-idx-2);
for (int expand=0; expand maxLength) {
maxStart = start;
maxLength = length;
}
}

return s.substr(maxStart, maxLength);
}
};
```

Reply ↓ **0**

vismaster
December 9, 2013 at 2:41 pm

first line should be:

#include

Reply ↓ **0**

illuminating
January 23, 2014 at 4:25 am

DP top down solution :

#include
#include
#include

```
using namespace std;

bool isPalindromString(const char * inputStr, int start, int end, int** palindromFlagTable){
if(start > end){
return false;
}
if(start == end){
palindromFlagTable[start][end] = 1;
return true;
}
if(palindromFlagTable[start][end] == 1 || palindromFlagTable[start][end] == -1){
return palindromFlagTable[start][end] == 1 ? true : false;
}
if(inputStr[start] != inputStr[end]){
palindromFlagTable[start][end] = -1;
return false;
}
if(end == start + 1){
palindromFlagTable[start][end] = 1;
return true;
}
bool ret = isPalindromString(inputStr, start + 1, end – 1, palindromFlagTable);
if(ret){
palindromFlagTable[start][end] = 1;
}else{
palindromFlagTable[start][end] = -1;
}
return ret;
}

int main(int argc, char* argv[]){

string input = string(“aaaabcbaacxxcaaccaa”);

const char* inputStr = input.c_str();
int strLen = strlen(inputStr);
int ** palindromFlagTable = new int*[strLen]();
for(int i = 0; i < strLen; i++){
palindromFlagTable[i] = new int[strLen]();
}

int maxLen = INT_MIN;
int begin = INT_MIN;
for(int i = 0; i i ; j –){
if(isPalindromString(inputStr, i , j, palindromFlagTable)){
```

```
if(j – i + 1 > maxLen){
maxLen = j – i + 1;
begin = i;
}
}
}
}

if(maxLen > 0){
printf(“max len is [%d], and begin is [%d] and the sub string is [%s]\n”, maxLen, begin, input.substr(begin, maxLen).c_str());
}else{
printf(“not found \n”);
}

for(int i = 0 ; i < strLen; i++){
delete[] palindromFlagTable[i];
}
delete[] palindromFlagTable;

}
```

Reply ↓ Report user **0**

meng yi
February 20, 2014 at 10:30 am

the second method is very interesting and simple!

Reply ↓ Report user **0**

mengfeng
March 8, 2014 at 7:40 pm

Python version:

```
import argparse

def extend_for_palindrome(string_input, index):
if index == 0 or index == len(string_input) -1 :
return string_input[0]
else:
span = 1
```

while index – span >=0 and index + span len(max_palindrome) else max_palindrome
return max_palindrome

if __name__ == '__main__':
parser = argparse.ArgumentParser(description='Find the longest palindrome')
parser.add_argument('-s', '–string', help='input string')

args = parser.parse_args()

string_input = args.string

print 'String input:', string_input

print 'Max Palindrome:', longest_palindrome(string_input)

Reply ↓ Report user +1

mengfeng
March 8, 2014 at 7:43 pm

Python version:

Re-posted with "code" tag:

```
import argparse

def extend_for_palindrome(string_input, index):
    if index == 0 or index == len(string_input) -1 :
        return string_input[0]
    else:
        span = 1
        while index - span >=0 and index + span  len(max_palindrome) else max_palindrome
    return max_palindrome

if __name__ == '__main__':
    parser = argparse.ArgumentParser( description='Find the longest palindrome')
    parser.add_argument( '-s', '--string', help='input string')

    args = parser.parse_args()

    string_input = args.string

    print 'String input:', string_input

    print 'Max Palindrome:', longest_palindrome(string_input)
```

Reply ↓ Report user 0

mengfeng
March 8, 2014 at 7:45 pm

Not sure why this function is missing after added code tag

```
def longest_palindrome(string_input):
max_palindrome = string_input[0]
for i, char in enumerate(string_input):
current_palindrome =extend_for_palindrome(string_input, i)
#print i, current_palindrome
max_palindrome = current_palindrome if len(current_palindrome) > len(max_palindrome)
else max_palindrome
return max_palindrome
```

Reply ↓ Report user 0

mengfeng
March 8, 2014 at 7:46 pm

```
def extend_for_palindrome(string_input, index):
    if index == 0 or index == len(string_input) -1 :
        return string_input[0]
    else:
        span = 1
        while index - span >=0 and index + span <= len(string_input) -1  and  string_input[inc
            span += 1

        return string_input[index - (span -1): index + (span -1 )+1]
```

Reply ↓ Report user 0

vikas
March 29, 2014 at 7:26 pm

your expand code doesnt for string “cabad”, it returns ab

Reply ↓ 0

Pingback: LeetCode - Longest Palindromic Substring | Darren's Blog

Krishna

April 23, 2014 at 3:40 am

Good Question, I have see this question first in this list and I was looking for solution, glad that I found a decent explanation here.

Reply ↓ 0

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

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

Jintao Xiao
August 8, 2014 at 11:54 pm

my method

```
public class Longest_palindrome_subsequence01 {
    public static int solution(String s) {
        if (s == null || s.equals("")) {
            return 0;
        }
        int longest_length = 0;
        char[] cc = s.toCharArray();
        StringBuilder lps = new StringBuilder();
        Stack sd = new Stack();
        int n = cc.length;
        int i = 0, j = n - 1;
        int tmp = n - 1;

        while (j > i) {
            if (cc[i] == cc[j]) {
                lps.append(cc[i]);
                sd.push(cc[i]);
                longest_length += 2;
                i++;
                j--;
            } else {
                j--;
            }

        }
        if (j == i) {
            longest_length++;
            lps.append(cc[i]);
        }
        while (!sd.empty()) {
            lps.append(sd.pop());
        }

        System.out.println(lps.toString());

        return longest_length;
    }
```

```
    <a href='http://leetcode.com/members/test/' rel="nofollow">@Test</a>
    public void testName() throws Exception {
        String s = "abacdfgdcaba";
        int a = solution(s);
        System.out.println(a);
    }
```

Reply ↓ Report user 0

Pingback: Longest Palindromic Substring [LeetCode 87] | CHazyhabiT

Pingback: Longest Palindromic Substring | akajj

Pingback: leetcode example | Multimedia Feedback Demo

Pingback: Algorithms – Strings and Numbers » Tech Blog

vinoth
November 2, 2014 at 10:16 pm

Thank you for the code. It really helped me. However, I found couple of issues with the code.
1. The return statement in “return s.substr(l+1, r-l-1);” must be changed to “return s.substr(l+1, r);”
2. String cannot be accessed by “s[l] == s[r]”, rather it must be s.charAt(l) == s.charAt(r)

Reply ↓ +4

Sean Jiang
November 5, 2014 at 7:30 am

Here is my solution. I think its an elegant O(N) solution in Java. Would like to be reviewed by the judge if possible.

```
public class MaxPalindromicSubString {
    String origs;
    String maxPalSub;  //Max Pal Sub String of DP sub solution for (i-1)
    String maxPalSuffix; //Max Pal Suffix, which potentially change the overall maxPalSub whe
    boolean isPalSuffixSame=true; //All same suffix will result in a potential PalSuffix chan

    public MaxPalindromicSubString(String input){
        this.origs = input;
    }

    public String compute(){
        maxPalSub=""+origs.charAt(0);
        maxPalSuffix=""+origs.charAt(0);
        isPalSuffixSame=true;
        for (int i=1;i=maxPalSub.length())
                {
                    maxPalSub = maxPalSuffix;
```

```
                }
            }
            else if (i-maxPalSuffix.length()-1>=0)
            {
                if (origs.charAt(i)==origs.charAt(i-maxPalSuffix.length()-1))
                {
                    maxPalSuffix = origs.charAt(i)+maxPalSuffix+origs.charAt(i);
                    if (origs.charAt(i)==origs.charAt(i-1) && isPalSuffixSame)
                    {
                        isPalSuffixSame=true; //Current suffix is still all same.
                    }
                    else
                    {
                        isPalSuffixSame=false;
                    }
                    if (maxPalSuffix.length()>=maxPalSub.length())
                    {
                        maxPalSub = maxPalSuffix;
                    }
                }
                else
                {
                    maxPalSuffix = ""+origs.charAt(i);
                    isPalSuffixSame = true; //One character is treated as all same
                }
            }
            else
            {
                maxPalSuffix=""+origs.charAt(i);
            }
        }
        return maxPalSub;
    }

    public static void main(String args[])
    {
        String input ="abcdcbbeddddebbcdefea";
        MaxPalindromicSubString mps = new MaxPalindromicSubString(input);
        System.out.println(mps.compute());

    }
}
```

Reply ↓ Report user 0

GeneralZyq
November 10, 2014 at 3:16 am

The third one is wonderful

Reply ↓ Report user 0

GaryZhu
January 15, 2015 at 8:04 am

```
public class LongestPalindrome {

    public static void main(String[] args) {
        String a = "1abaaba3";
        System.out.println(getLongestPalindrome(a));
    }

    public static String getLongestPalindrome(String input) {
        byte[] a = input.getBytes();
        String result = "";
        for (int i = 0; i < a.length; i++) {
            for (int j = i + 1; j <a> (j - i) ? result : input
                                .substring(i, j + 1);
                    }
                }
            }
        }

        return result;
    }

    public static boolean isPalindrome(String str) {
        Stack stack = new Stack();

        for (int i = 0; i < str.length(); i++) {
            if (i < str.length() / 2) {
                stack.add(str.charAt(i));
            } else {
                if (i == str.length() / 2 && str.length() % 2 != 0) {
                    continue;
                }
                char temChar = (char) stack.pop();
                if (str.charAt(i) != temChar) {
                    return false;
                }
            }
        }

        if (stack.size() == 0)
            return true;
        else
            return false;

    }

}
```

Reply ↓ Report user 0

GaryZhu

January 15, 2015 at 8:08 am

```
public class LongestPalindrome {

    public static void main(String[] args) {
        String a = "1abaaba3";
```

```
        System.out.println(getLongestPalindrome(a));
    }

    public static String getLongestPalindrome(String input) {
        byte[] a = input.getBytes();
        String result = "";
        for (int i = 0; i < a.length; i++) {
            for (int j = i + 1; j <a> (j - i) ? result : input
                                .substring(i, j + 1);
                    }
                }
            }
        }

        return result;
    }

    public static boolean isPalindrome(String str) {
        Stack stack = new Stack();

        for (int i = 0; i < str.length(); i++) {
            if (i < str.length() / 2) {
                stack.add(str.charAt(i));
            } else {
                if (i == str.length() / 2 && str.length() % 2 != 0) {
                    continue;
                }
                char temChar = (char) stack.pop();
                if (str.charAt(i) != temChar) {
                    return false;
                }
            }
        }

        if (stack.size() == 0)
            return true;
        else
            return false;

    }

}
```

Reply ↓ Report user 0

Pingback: Longest Palindromic Substring [LeetCode 87] - Uzumaki Kyuubi

Pingback: Longest Palindromic Substring [LeetCode 5] - Uzumaki Kyuubi

Truong Khanh
February 24, 2015 at 9:08 pm

I also discuss this problem at my blog http://www.capacode.com/?p=72

Reply ↓ Report user +1

Yat Kam
May 17, 2015 at 1:56 am

java version

```
public static String longestPalindrome(String s) {
        String maxStr = "";
        for(int i = s.length();i >= 0;i --){
            maxStr = isPalindrom(s, i);
            if(maxStr.length() !=0)
                break;
        }
        return maxStr;
    }

    public static String isPalindrom(String s,int len){
        int i = 0;
        StringBuffer str = new StringBuffer(s);
        String str1;
        String str2;
        String str3 = "";
        while(i + len <= s.length()){
            str1 = str.substring(i,i + len).toString();
            StringBuffer str4 = new StringBuffer(str1);
            str2 = str4.reverse().toString();
            if(str1.equals(str2) && str3.length() < str1.length()){
                str3 = str1;
                break;
            }
            else{
                i ++;
            }
        }
        return str3;
    }
```

Reply ↓ +1

Pingback: 「LeetCode」试题总结 – 回文数汇总 | Nillhex

XiangyangZhang
June 15, 2015 at 6:15 pm

if we import the string “character”, the second method exports “ara”,
However, the right answer may be “carac”.
So the second method may be wrong.

Reply ↓ +1

Pingback: Find the longest pallindrome substring - BlogoSfera

รับซื้อ Patek Philippe
October 5, 2015 at 9:26 pm

Quality articles is the crucial to invite the viewers to pay a quick visit the web site, that's what this website is providing.

Reply ↓ **0**

Haoyu
October 8, 2015 at 4:20 pm

The last O(n^2) expand center method, while (l >= 0 && r <= n-1 && s[l] == s[r]) the s is still string not array .

Reply ↓ **0**

Pingback: LeetCode 5: Longest Palindromic Substring (Medium) | helloyuan

Pingback: 5 Longest Palindromic Substring | The Beauty of Programming

Pingback: A few good programming questions – THE INTERVIEW HACKER

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