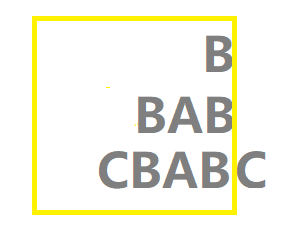
There is another algorithms rather than manancher's algorithms for solve max palindromic substring in linear time, which I came up with the idea in about 2014, for a question about medium level of difficulty in hackerranker. Because I think this solution is simple, but few articles, and pagers can be found via google, youtube.com, etc., so I decide to share this solution to GeeksforGeeks.

## 1. simple explanation of palindromic string

For example string aba, abcba, goodoog, bab, and geekskeeg, if you reverse these strings, they are still same with the origin strings.

## 2. solution to solve max palindromic substring

For example for string CBABC, you want to find longest palindromic substring for it.  For assumption, when you find substring from left to right, character by character. Till the last second character B, you have find the longest substring till B is BAB. The second longest plaindromic substring till B is B. we link palindromic sustrings from longest to shortest to a link. You can base  on this result  to find  palindromic substrings till next character C. obviously  longest palindromic substring till C is CBABC, and the second longest substring till C is C. You can use this  link to find another next link  too,if have another character after C.



## 3. source code in java

Java

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class PalindromicString{

public static List<String>distinctPalindrome=new ArrayList<String>();

public static class Node {

int start, end;

Node link;

Node[] childrens = new Node[26];

int len;

public Node(int start, int end) {

this.start = start;

this.end = end;

this.len = this.end - this.start + 1;

}

public Node(int len) {

this.len = len;

}

};

static Node root1 = new Node(-1);

static Node root2 = new Node(0);

static {

root2.link = root1;

}

static Node cn = root2;

static int max = 0;s

public static void insert(char[] cs, int idx) {

int ci = cs[idx] - 97;

Node lastLnode = cn;

Node lastCn = null;

do {

if (lastLnode == root1) {

if (root1.childrens[ci] == null) {

root1.childrens[ci] = new Node(idx, idx);

distinctPalindrome.add(new String(cs,idx,1));

root1.childrens[ci].link = root2;

}

if (lastCn == null) {

cn = root1.childrens[ci];

} else {

lastCn.link = root1.childrens[ci];

}

break;

} else {

int start = idx - lastLnode.len - 1;

int end = idx;

if (start >= 0 && cs[start] == cs[end]) {

if (lastLnode.childrens[ci] != null) {

if (lastCn == null) {

cn = lastLnode.childrens[ci];

} else {

lastCn.link = lastLnode.childrens[ci];

}

break;

} else {

lastLnode.childrens[ci] = new Node(start, end);

distinctPalindrome.add(new String(cs,start,end-start+1));

if (lastCn == null) {

cn = lastLnode.childrens[ci];;

} else {

lastCn.link = lastLnode.childrens[ci];

}

lastCn = lastLnode.childrens[ci];

}

}

lastLnode = lastLnode.link;

}

} while (true);

}

public static void recalMax(int len) {

max = max > len ? max : len;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

char[] cs = scanner.nextLine().toCharArray();

for (int i = 0; i < cs.length; i++) {

insert(cs, i);

recalMax(cn.len);

}

System.out.println(max);

for(String p:distinctPalindrome) {

System.out.println(p);

}

scanner.close();

}

}

## 4. time complexity

The time complexity depend on and length of link and the track to the next  palindromic substrings whether have corresponding  substring, it works like cache, whether have a hit, if have a hit, less back track, less time. but we can just consider the over all length of all palindromic substring links is enough to charge this solution is no more than n square, first of all, all the length of link is  not more than half of n square.

1. ***if all charcters are same,length of links is half of n square, 2 tries one hit, all hit rate is 50%, time complexicity is O(2n) that is O(n);***
2. ***more different charcters the link is shorter, the hit rate is lower. in the contrary,more length of link, higher rate to the hit;***
3. ***if meet new character, the  length of new link is decreased to nearly one, reduce the top bound of next track back;***
4. ***if more track back, take more time, but next link usually is shorter, reduce the top bound of next track back;***
5. ***next link is at most one longer than last link.***

Over all, time complexity is optimistic. for the time complexcity, I haven't come up with a conclusive formula to calculate it. if you are interested , you are open to provide your method or formula.

## 5.this solution VS manancher's algorithms

|  |  |  |  |
| --- | --- | --- | --- |
|  | ****this solution**** | ****manancher's algorithms**** |  |
| time complexity | almost linear O(N) | linear O(N) |  |
| space complexity | N | N |  |
| can find all distinct palindromic substring | Yes | No,need extra time and space, many other source code |  |
|  |  |  |  |