Tutorial

Practice

Tutorial on Manachar's Algorithm

AUTH
omar khaled abdelaziz abdelnabi

Manacher's Algorithm has one single application. It is used to find the Longest Palindromic Sub-string in any string. This algorithm is required to solve sub-problems of some very hard problems. This article explains the basic brute force method first and then moves on to explain the optimized Manacher's Algorithm.

Brute Force Approach:

Find all possible sub-strings using 2 nested loops, this solution has $O(N^2)$ where N is the length of the given string. Then for every substring, check if it is a palindrome or not in $O(N)$, so the total time complexity is $O(N^3)$.

This solution could be improved by selecting every letter in the given string as the center $O(N)$, then find the longest palindromic substring around this center $O(N)$, so the total time complexity is $O(N^2)$. For

example: given string is "$czbza$", when b is selected as the center, it produces the longest palindromic substring "zbz".

However, $O(N^2)$ solution could be improved using some clever observations. Following is the optimized solution.

Manacher's Algorithm

Assume the given String S has a length of N, S = "$abababa$". Create a string Q of length $2 \cdot N + 1$, by inserting any letter that doesn't appear in the input string (call it special character for the purpose of this article), in the spaces between any 2 characters. Also, insert this special character in the beginning and the end of the string. If "#" is chosen as special character then the new string Q would look like "#a#b#a#b#a#b#a#".

Calculate the longest palindromic substring in each center. Expand around each character i in the new string, then store the number of letters, in the longest palindromic substring with character i as the center, divided by 2. The stored number is divided by 2 because the palindromic substring has it's 2 same parts around the center.

Above process would yield an array
$P = [0, 1, 0, 3, 0, 5, 0, 7, 0, 5, 0, 3, 0, 1, 0]$. Each number m, in the array P, indicates that there are m corresponding letters in both sides around a center i. So the palindromic substring = m letters in the left side + center + m letters in right side.

Observation (1):

For the center index $c = 7$ i.e $P[c] = 7$ which has the longest palindromic substring. Notice that the numbers in array P after center $c = 7$ are same as numbers before center c, so avoid expanding around all letters after center c, however just put their values directly using the Mirror (by copying the first half of array P in its other half) property.

Observation (2):

Unfortunately, Mirror property can't be applied in all cases. For example: S = "$acncacn$", the new string
Q = "#a#c#n#c#a#c#n#".
The result array P = $[0, 1, 0, 1, 0, 5, 0, 1, 0, 5, 0, 1, 0, 1, 0]$.

Consider the center $c = 5$. The mirror property applies in $P[4] = p[6], p[3] = p[7], p[2] = p[8]$. But why $p[1] \; ! = \; p[9]$? So Mirror property doesn't work in all cases, because in this case there is another palindrome with center $c = 9$. This new palindrome, with center $c = 9$, goes beyond the limits of the first palindrome with center $c = 5$.

Algorithm Steps:

Let the 2 limits of the first palindrome with center c: a left limit l, a right limit r. l, r have references over the last 2 corresponding letters in the palindrome sub-string. A letter w with index i in a palindrome substring has a corresponding letter w' with index i' such that the $c - i$ = $i' - c$.

(1) If($p[i] \leq r - i'$),

So $p[i'] = p[i]$ which means that palindrome with center i' can't go beyond the original palindrome, so apply the Mirror Property directly.

(2) Else $p[i'] \geq p[i]$,

This means that palindrome with center i' goes beyond the original palindrome, so expanding around this center i' is needed.

Let $d = \ distance\ r - i'$, so expanding around center i' will start from $(i' - d) - 1$ with $(i' + d) + 1 = (r + 1)$ and so on... because the interval $[i' - d : i' + d]$ is already contained in the palindrome with center i'.

The algorithm has 2 nested loops, outer loop check if there will be an expanding around current letter or not. This loop takes N steps.
Inner loop will be used in case of expanding around a letter, but it is guaranteed that it takes at most N steps by using the above 2 observations.
So the total time = $2 \cdot N = O(N)$.

(3) Update c, r when a palindrome with center i' goes beyond the original palindrome with center c.
$c = i', r = i' + p[i']$ as the next expanding will be around center i'.

Note: Insert another 2 different special characters in the front and the

end of string Q to avoid bound checking.

Implementation:

```
#include <bits/stdc++.h>
using namespace std;
#define SIZE 100000 + 1

int P[SIZE * 2];

// Transform S into new string with special characters inse
string convertToNewString(const string &s) {
    string newString = "@";

    for (int i = 0; i < s.size(); i++) {
        newString += "#" + s.substr(i, 1);
    }

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

string longestPalindromeSubstring(const string &s) {
    string Q = convertToNewString(s);
    int c = 0, r = 0;                // current center, rig

    for (int i = 1; i < Q.size() - 1; i++) {
        // find the corresponding letter in the palidrome s
        int iMirror = c - (i - c);

        if(r > i) {
            P[i] = min(r - i, P[iMirror]);
        }
```

```
        // expanding around center i
        while (Q[i + 1 + P[i]] == Q[i - 1 - P[i]]){
            P[i]++;
        }

        // Update c,r in case if the palindrome centered at
        if (i + P[i] > r) {
            c = i;              // next center = i
            r = i + P[i];
        }
    }

    // Find the longest palindrome length in p.

    int maxPalindrome = 0;
    int centerIndex = 0;

    for (int i = 1; i < Q.size() - 1; i++) {

        if (P[i] > maxPalindrome) {
            maxPalindrome = P[i];
            centerIndex = i;
        }
    }

    cout << maxPalindrome << "\n";
    return s.substr( (centerIndex - 1 - maxPalindrome) / 2,
}

int main() {
    string s = "kiomaramol\n";
    cout << longestPalindromeSubstring(s);
    return 0;
}
```

TEST YOUR UNDERSTANDING

Longest Palindromic String

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

Input:
First and only line will contain string S.

Output:
Print the length of the longest palindrome substring in the first line. In the second line print the longest palindromic substring in S. If there is more than one palindromic substring with the maximum length, output the first one.

Constraints:
$1 \leq N \leq 10^5$
String S will only contain lower case English alphabet $[a - z]$.

SAMPLE INPUT

```
czcaba
```

SAMPLE OUTPUT

```
3
czc
```

Enter your code or Upload your code as file.

All changes saved

Visual Basic (mono vbnc 4.0.1)

```
1  Loading...
```

1:1

COMPILE & RUN SUBMIT

Press ctrl-space for autocomplete suggestions. POWERED BY codetable

ABOUT US

- Blog
- Engineering Blog
- Updates & Releases
- Team
- Careers

HACKEREARTH

- API
- Chrome Extension
- CodeTable
- HackerEarth Academy

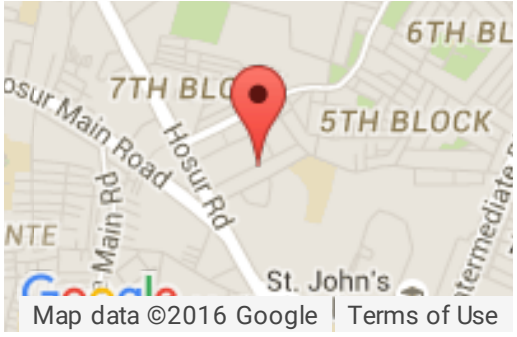
DEVELOPERS

- AMA
- Code Monk
- Judge Environment
- Solution Guide
- Problem Setter Guide

EMPLOYERS

- Developer Sourcing
- Lateral Hiring
- Campus Hiring
- Hackathons
- FAQs

REACH US



IIIrd Floor,
Salarpuria Business Center,
4th B Cross Road,
5th A Block,
Koramangala

Industrial Layout,
Bangalore,
Karnataka 560095,
India.

contact@hackerear
+91-80-4155-4695
+1-650-461-4192