Manacher's Algorithm

Authored by HackerRank

Manacher's algorithm is a very handy algorithm with a short implementation that can make many programming tasks, such as finding the number of palindromic substrings or finding the longest palindromic substring, very easy and efficient. The running time of Manacher's algorithm is $O(N)$ where N is the length of the input string.

Let us assume we are given a string S with a length of N. The objective of this algorithm is to build a table $P[]$ such that knowing the value of $P[i]$ enables us to find the length of the longest palindromic substring with center at position i for all $1 \leq i \leq N$.

Suppose $S =$ **"abba"** and consider 1-based indexing of the string. Then the palindromic substring **"bb"** may have an ambiguity in terms of its center as index 2 and index 3 are both its centers. To avoid this ambiguity, we insert an arbitrary character (that doesn't appear in the string), say **'#'**, between every two characters of S and in the beginning and end of S.

So, the string now transforms to $T =$ **"#a#b#b#a#"**. Notice that every palindromic substring in the original string, regardless of length, corresponds to a palindromic substring in the transformed string of *odd* length, so we can now define $P[i]$ to mark the radius of the largest *odd*-length palindromic substring centered at index i.

Suppose R is the rightmost boundary of the longest palindrome centered at i. Then $P[i] = R - i$. $(1 \leq i \leq 2N - 1)$

Thus, the length of the longest palindromic substring for every index in the original string can be easily recovered.

Let us see how the P values will be filled for the given string S.

```
i = 1 2 3 4 5 6 7 8 9

T = # a # b # b # a #

P = 0 1 0 1 4 1 0 1 0
```

From the $P[]$ table we can see that the length of the longest palindromic substring in S is 4.

Now the hardest and the most essential part of this algorithm is the calculation of the P table.

Let T be the string formed after inserting characters **'#'** between the characters of string S.

Let C be the center of the palindrome currently known to include the boundary closest to the right end of our string T.

Go to Top

Let R be the rightmost boundary of the palindrome centered at C.

Let i be the position of an element in T whose palindromic span is being determined, with i always to the right of C.

Let i' be the mirrored position of i with respect to C. It can be seen that

$$i' = C - (i - C) = 2C - i.$$

We will iterate i from 1 to the length of T and determine $P[i]$ assuming that $P[i']$ for all $i' < i$ has already been calculated.

Consider we are given a string $S =$ **"babcbabcbaccba"**.

Thus, we have $T =$ **"#b#a#b#c#b#a#b#c#b#a#c#c#b#a#"** after inserting **'#'** between every two characters.

Consider the following three cases:

Case 1: The length of the longest palindrome centered at i' is such that the left boundary of the palindrome does not extend beyond or until the left boundary of the longest palindrome centered at C.

Let the left boundary of palindrome at C be L . We know that $T[C+k] = T[C-k]$ for all $k < R - C$ since the substring of T centered at C and radius $R - C$ is a palindrome. *(1)*

Notice that $P[X]$ gives the number of odd length palindromes centred at X and hence $2P[X] - 1$ is the length of the largest palindrome centred at X.

So consider $T[i' - k]$ and $T[i + k]$ for all $k \leq P[i']$.

$k \leq P[i']$ implies $k < i' - L$ as the subpalindrome centred at i' has a left boundary to the right of the left boundary of the palindrome centred at C.

As i' is a mirror of i about C,

$$k < R - i$$

.

$$i' = 2C - i$$

$$T[i' - k] = T[2C - i - k] = T[C - (i + k - C)]$$

$$T[i + k] = T[C + (i + k - C)]$$

$$i + k - C < i + R - i - C$$

$$i + k - C < R - C$$

Hence

$$T[i' - k] = T[C - k']$$

$$T[i + k] = T[C + k']$$

where

$$k' < R - C$$

Hence from statement *(1)*

$$T[i + k] = T[i' - k]$$

for all

$$k < i' - L$$

.

Hence $P[i] = P[i']$ for this case.

Also as the right boundary of the palindrome centred at i does not exceed R we should not update the R and C value.

Case 2:

Consider that the palindrome centred at i' extends beyond the left boundary of the palindrome centred at C.

Consider $T[L - 1]$.

$\mathbf{mirror}(L - 1)$ w.r.t i'

$$= M_1 = 2i' - (L - 1)$$

$$T[L - 1] = T[M_1]$$

$\mathbf{mirror}(M_1)$ w.r.t C

$$= M_2 = 2C - M_1$$

$$T[M_1] = T[M_2]$$

$\mathbf{mirror}(M_2)$ w.r.t i

$$= M_3 = 2i - M_2 = 2i - 2C + M_1$$

$$= 2i - 2C + 2i' - L + 1 = 2(i - C + i') - L + 1$$

$$= 2(2C - i' - C + i') - L + 1 = (2C - L) + 1 = R + 1$$

Now suppose $T[M_3] = T[M_2]$.

Hence

$$T[M_3] = T[M_1] = T[L - 1].$$

Then $T[R + 1] = T[L - 1]$ which is not possible since it would mean that it is possible to extend the palindrome centred at C.

Therefore,

$$T[M_3] \neq T[M_2]$$

and it is not possible to extend the palindrome centred at i beyond R.

Using the analysis of *Case 1* we can say that the radius of the palindrome centred at i is equal to the distance from the right boundary of i.

Hence

$$P[i] = R - i + 1.$$

Case 3:

In this case, The left boundary of the palindrome centred at i' is the same as the left boundary of the palindrome centred at C.

Using the math mentioned in *Case 1* it can be proved that the right boundary of the palindrome centred at i can be extended atleast until R.

So

$$P[i] \geq R - i + 1.$$

However there is no inequality that ascertains that we cannot extend the right boundary beyond that . So we try to extend the right boundary of i.

```
int curR = R;
while (T[curR] == T[mirror_about_i(curR)]) {
    curR++;
}
R = curR;
P[i] = curR - i;
C = i;
```

Here is the code for Manacher's algorithm:

```
int C = 0, R = -1, rad;
for (int i = 0; i < T.length(); ++i) {
    if (i <= R) {
        rad = min(P[2*C-i], R-i);
    } else {
        rad = 0;
    }
    // Try to extend
    while (i+rad < T.length() && i-rad >= 0 && T[i-rad] == T[i+rad]) {
        rad++;
    }
    P[i] = rad;
    if (i + rad - 1 > R) {
        C = i;
        R = i + rad - 1;
    }
}
```

The time taken for the algorithm is $O(N)$ where N is the length of T.