<https://leetcode.com/problems/shortest-palindrome/description/>

You are given a string s. You can convert s to a palindrome by adding characters in front of it.

Return *the shortest palindrome you can find by performing this transformation*.

**Example 1:**

Input: s = "aacecaaa"

Output: "aaacecaaa"

**Example 2:**

Input: s = "abcd"

Output: "dcbabcd"

**Constraints:**

* 0 <= s.length <= 5 \* 104
* s consists of lowercase English letters only.

**Attempt 1: 2023-07-18**

**Solution 1:  Brute Force (10min, TLE 120/123)**

**'We can convert it to an alternative problem"find the longest palindrome substring starts from index 0".'**

class Solution {

public String shortestPalindrome(String s) {

// Set as -1 is convenient for no palindrome found and whole

// string has to be considered as compensation to be added

// in front of s, work together with s.substring(-1 + 1)

int cutpoint = -1;

for(int i = s.length() - 1; i >= 0; i--) {

if(isPalindrome(s, 0, i)) {

cutpoint = i;

break;

}

}

// 'cutpoint + 1' means the substring should start from the

// 1st character after cutpoint

StringBuffer sb = new StringBuffer(s.substring(cutpoint + 1));

return sb.reverse().toString() + s;

}

private boolean isPalindrome(String s, int i, int j) {

while(i <= j) {

if(s.charAt(i) == s.charAt(j)) {

i++;

j--;

} else {

return false;

}

}

return true;

}

}

**Solution 2:  Rolling Hash (360 min)**

**Wrong Solution**

**We cannot scan backward as for(int i = n - 1; i >= 0; i--) like what we did in Brute Force solution, because not like Brute Force, we can do backward traverse and try to identify a largest i if s[0, i] is a palindrome, if cutpoint 'i' more close to last position, it will find more quickly than scanning forward.**

**But here is totally different, in Rolling Hash, we have to maintain an iterative way to calculate a 'multiplier' kind of 'forward\_hash' as what we did in L1044, we deal with each digit individually, not like Brute Force treat as a substring as s[0, i], e.g if we scan backward, the last digit in "abcd" as 'd' will be treated individually and result is 'd' has same forward & backward hash, then we wrongly update 'cutpoint' from -1 to 3, even "abcd" is not a palindrome at all.**

**Hence we have to scan from beginning like how we di Rolling Harsh usually, gradually increment a new digit on scanned substring, then calculate a new forward & backward hash**

class Solution {

public String shortestPalindrome(String s) {

int n = s.length();

int cutpoint = -1;

long mod = 1000000007;

int base = 256;

long pow = 1;

long forward\_hash = 0;

long backward\_hash = 0;

for(int i = n - 1; i >= 0; i--) {

forward\_hash = (forward\_hash \* base + s.charAt(i)) % mod;

backward\_hash = (backward\_hash + s.charAt(i) \* pow) % mod;

pow = pow \* base % mod;

if(forward\_hash == backward\_hash) {

cutpoint = i;

break;

}

}

return new StringBuilder().append(s.substring(cutpoint)).reverse().append(s).toString();

}

}

**Note: Compare to L1044, the Rolling Hash in L214 is more like calculate the 'multipler' in L1044, since we don't need to build a sliding window to find same substring, instead we just need to add one character in each loop iteration forward and backward**

class Solution {

public String shortestPalindrome(String s) {

int n = s.length();

int cutpoint = -1;

long mod = 1000000007;

int base = 256;

long pow = 1;

long forward\_hash = 0;

long backward\_hash = 0;

for(int i = n - 1; i >= 0; i--) {

forward\_hash = (forward\_hash \* base + s.charAt(i)) % mod;

backward\_hash = (backward\_hash + s.charAt(i) \* pow) % mod;

pow = pow \* base % mod;

if(forward\_hash == backward\_hash) {

cutpoint = i;

break;

}

}

return new StringBuilder().append(s.substring(cutpoint)).reverse().append(s).toString();

}

}

**Refer to**

<https://leetcode.com/problems/shortest-palindrome/solutions/60153/8-line-o-n-method-using-rabin-karp-rolling-hash/>

This problem is indeed computing the longest palindromic prefix of a string s. A naive approach would be computing all the prefixes of s and its reverse, and then finding the longest pair of prefixes that are equal.

Unfortunately, this method requires quadratic time and space since the length sum of all prefixes is 1+2+...+|s| = Θ(|s|^2).

Via the help of the Rolling Hash method, the above process can be optimized down to linear time. For more details, you can visit [here](https://en.wikipedia.org/wiki/Rolling_hash) and [here](http://courses.csail.mit.edu/6.006/spring11/rec/rec06.pdf).

public String shortestPalindrome(String s) {

int n = s.length(), pos = -1;

long B = 29, MOD = 1000000007, POW = 1, hash1 = 0, hash2 = 0;

for (int i = 0; i < n; i++, POW = POW \* B % MOD) {

hash1 = (hash1 \* B + s.charAt(i) - 'a' + 1) % MOD;

hash2 = (hash2 + (s.charAt(i) - 'a' + 1) \* POW) % MOD;

if (hash1 == hash2) pos = i;

}

return new StringBuilder().append(s.substring(pos + 1, n)).reverse().append(s).toString();

}

<https://leetcode.com/problems/shortest-palindrome/solutions/60153/8-line-o-n-method-using-rabin-karp-rolling-hash/comments/61282>

Consider a decimal example (base = 10). Say we are given a number 7134. If we read it from left to right, we get 7134. And 4317 if we read it from right to left.

hash1 is the left--to-right fashion:

* hash1 = 0
* hash1 = 0 \* 10 + 7 = 7
* hash1 = 7 \* 10 + 1 = 71
* hash1 = 71 \* 10 + 3 = 713
* hash1 = 713 \* 10 + 4 = 7134

hash2 is the right-to-left fashion:

* hash2 = 0
* hash2 = 0 + 7 \* 1 = 7
* hash2 = 7 + 1 \* 10 = 17
* hash2 = 17 + 3 \* 100 = 317
* hash2 = 317 + 4 \* 1000 = 4317

A palindrome must be read the same from left to right and from right to left. So in this case, 7134 is not a palindrome.

**Above is an example for the decimal case, and for rolling hashing, the only differences are:**

1. **Base is not 10, but any constant >= 26.**
2. **hash1 and hash2 are not the exact value, but the exact value modulo a big prime. (Since the exact value is too large to fit in a 32-bit integer.)**

As you may notice, the rolling hash function is not an injection, which means that two different strings may share the same hash code.(This is also commonly called a conflict.) But in real, it is very difficult to trigger many conflicts (sometimes not even a single one) unless there are sufficiently many strings given. Therefore, if hash1 is not equal to hash2 for some string, then definitely it is not a palindrome. On the other hand, if they are equal, it means the string is a palindrome with extreme high probability.