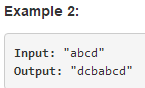
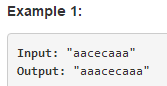
Shortest Palindrome最短回文问题

**palindrome** 英[ˈpælɪndrəʊm] 美[ˈpælɪndroʊm] n. 回文（指顺读和倒读都一样的词语）;

[例句]It's not a palindrome. 数组不是回文的了。

# Shortest Palindrome最短回文

Given a string s, you are allowed to convert it to a palindrome **by adding characters in front of it.** Find and return the shortest palindrome you can find by performing this transformation.



注意：这里只要求在字符串的前面添加字符，因此只需要找出前面满足回文的最大字符串，然后补充后面多余的字符串就可以。

**算法代码**：

class Solution {

public String shortestPalindrome(String s) {

if(s == null || "".equals(s)||s.length() == 1){

return s;

}

char[] chars = s.toCharArray();//将字符串转换为字符数组，可以提高效率

int len = chars.length;

int end = len -1;

for(;end > 0;end --){

//是否是回文

**if(isPalindrome(chars,0,end)){**

**break;**

**}**

}

//将不满足回文的那部分字符添加上

StringBuilder sb = new StringBuilder();

for(int i = len -1;i > end;i--){

sb.append(chars[i]);

}

sb.append(s);

return sb.toString();

}

//判断字符数组中某段是否是回文

private boolean isPalindrome(char[] chars,int start,int end){

while(start < end){

if(chars[start] != chars[end]){

return false;

}

start++;

end--;

}

return true;

}

}

**高效率算法**：(看不懂)如果不是很有必要就不看了。

class Solution {

public String shortestPalindrome(String s) {

char[] ca = s.toCharArray();

int n = ca.length;

/\*if (isPalindrome(ca, 0, n-1)) return s;

StringBuilder sb = new StringBuilder();

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

sb.append(ca[i]);

if (isPalindrome(ca, 0, i-1))

return sb.append(s).toString();

}

return sb.append(s).toString();\*/

int idx = 0;

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

if (ca[idx] == ca[j])

idx++;

}

if (idx == n) return s;

String sub = s.substring(idx);

StringBuilder sb = new StringBuilder(sub);

return sb.reverse().toString() + shortestPalindrome(s.substring(0, idx)) + sub;

//return sb.reverse().toString() + s.substring(0, idx) + sub;

}

/\*public boolean isPalindrome(char[] s, int start, int end) {

while (start < end)

if (s[start++] != s[end--]) return false;

return true;

}\*/

}

# Palindrome Number 回文数字

Determine whether an integer is a **palindrome**. An integer is a palindrome when it reads the same backward as forward.

**Example 1**: Input: 121 Output: true

**Example 2**: Input: -121 Output: false

Explanation: From left to right, it reads -121. From right to left, it becomes 121-. Therefore it is not a palindrome.

**Example 3**: Input: 10 Output: false

Explanation: Reads 01 from right to left. Therefore it is not a palindrome.

Follow up: Coud you solve it without converting the integer to a string?

算法1：首先写一个转成字符串后的算法。

class Solution {

public boolean isPalindrome(int x) {

if(x < 0){

return false;

}

**char[] nums = (""+ x).toCharArray();**

int left = 0;

int right = nums.length -1;

**while(left < right){**

**if(nums[left++] != nums[right--]){**

**return false;**

**}**

**}**

return true;

}

}

算法2：

// without converting the integer to a string

class Solution {

public boolean isPalindrome(int x) {

if(x < 0) { return false;}

if(x == 0){ return true; }

//首先获取x的最大位数，从两个方向获取各个位上的值，

// 一是从高到低位(这个需要先寻找最高位)，一是从低到高位(不断求余即可)

//寻找最高位

int div = 1;

while(x/div >= 10){//条件很关键x/div != 0这个就是错误的，因为

div = div\*10;

}

// while(x/div != 0){//条件很关键x/div != 0这个在输入1000000001时是错误的，因为div就超过了int的范围不要先多10倍，又降下来

// div = div\*10;

// }

// div = div/10;

**while(x != 0){**

**if(x/div != x%10) return false;**

**x = (x%div)/10;**

**div /= 100;**

**}**

return true;

}

}

算法3：**将数字转换成逆过来的数字**，如1234转换为4321，判断是否相等，若相等则是回文，否则不是。(**这种算法有可能导致溢出。**)

// without converting the integer to a string

class Solution {

public boolean isPalindrome(int x) {

if(x < 0) return false;

if(x == 0) return true;

**//转换成逆过来的数字**

int rev = 0;

int xx = x;

while(xx != 0){

rev = rev\*10 + xx%10;

xx = xx/10;

}

return rev==x;

}

}