34 寻找最近的回文数

更新时间：2019-09-24 09:38:29

> 梦想只要能持久，就能成为现实。我们不就是生活在梦想中的吗？
>
> ——丁尼生

刷题内容

> 难度: **Hard**
>
> 题目链接：https://leetcode-cn.com/problems/find-the-closest-palindrome/submissions/

题目描述

给定一个整数 n，你需要找到与它最近的回文数（不包括自身）。

“最近的”定义为两个整数差的绝对值最小。

示例 1:

输入: "123"
输出: "121"
注意:

n 是由字符串表示的正整数，其长度不超过18。
如果有多个结果，返回最小的那个。

解题方案

思路1 时间复杂度: O(len(N)) 空间复杂度: O(1)

做这道题的时候，我各种冥思苦想啊，太难啦！后面学统计的女朋友来了，看完题目几秒钟后她就给出了解法，太厉害了！

总共三种可能性，我们把n的字符形式按照长度切成两半，前半部分和后半部分，靠的最近的回文数字res无非就是：
- res的前半部分：{n的前半部分+1，n的前半部分-1, n的前半部分自身} 中选一个
- res的后半部分：res的前半部分的反转
- res前半部分和res后半部分贴在一起就是我们最终的结果res

另外需要注意前半部分+1可能会进位，比如说n=999，那么99+1 = 100， 那么我们会想要100 - 001吗？我们肯定会要1001呀，因为用前面那种一下子进了2位，肯定没有进一位更近，格式为`'1' + '0' * (len(n_str) - 1) + '1'`；前半部分-1可能会减位，同理格式是`'9' * (len(n_str) - 1)`

但是这个代码有两个corner case没考虑到，那就是n为10和11的时候我们都需要返回9而不是"11"和“00"

因为我们只关注了n的长度，所以时间复杂度为O(len((n)))

python beats 100%

```
class Solution:
    def nearestPalindromic(self, n: str) -> str:
        if 9 < int(n) < 12:
            return '9'
        r_half_len = len(n) // 2  # res后半部分的长度
        if len(n) & 1 == 0:  # n长度为偶数
            num_digits = len(n) // 2  # n前半部分长度
            l_half = n[:num_digits]  # n前半部分
        else:  # n长度为奇数
            num_digits = (len(n) + 1) // 2
            l_half = n[:num_digits]  # n前半部分长度

        # 第一种情况：(n的前半部分+1) + reversed(n的前半部分+1)
        if len(str(int(l_half) + 1)) > num_digits:  # 进位了
            candidate1 = '1' + '0' * (len(n) - 1) + '1'  # 后半部分
        else:
            candidate1 = str(int(l_half) + 1) + str(int(l_half) + 1)[:r_half_len][::-1]

        # 第二种情况：(n的前半部分-1) + reversed(n的前半部分-1)
        if len(str(int(l_half) - 1)) < num_digits:  # 减位了
            candidate2 = '9' * (len(n) - 1)
        else:
            candidate2 = str(int(l_half) - 1) + str(int(l_half) - 1)[:r_half_len][::-1]

        # 第三种情况：n的前半部分 + reversed(n的前半部分)
        candidate3 = str(int(l_half)) + str(int(l_half))[:r_half_len][::-1]

        # 因为题目说了如果有多个结果，返回最小的那个，所以我们需要这样把小的数字放在前面
        candidates = [candidate2, candidate3, candidate1]
        if candidate3 == n[::-1]:  # 题目说了不能选自身，所以这种情况应该删除
            candidates.remove(candidate3)

        candidates.sort(key=lambda x: abs(int(x) - int(n)))
        return candidates[0]
```

c++ beats 100%

```
class Solution {
public:
    //n位开头转成回文数，odd表示是否奇数位
```

```
//n位开头转成回文数，odd表示是否奇数位
string tranToPalindromic(string n, bool odd) {
    string other = "";
    if (odd) {
        for (int i = n.size() - 2;i >= 0;i--) {
            other.push_back(n[i]);
        }
    } else {
        for (int i = n.size() - 1;i >= 0;i--) {
            other.push_back(n[i]);
        }
    }
    return n + other;
}
//将字符串倒过来
string transpose(string s) {
    string ret = "";
    for (int i = s.size() - 1;i >= 0;i--) {
        ret.push_back(s[i]);
    }
    return ret;
}
//将数字转成字符串
string itoa(long long n) {
    string ret = "";
    if (n == 0) {
        return "0";
    }
    while (n != 0) {
        ret.push_back('0' + (n % 10));
        n /= 10;
    }
    return transpose(ret);
}
//将字符串转成数字
long long atoi(string s) {
    long long ret = 0;
    for (int i = 0;i < s.size();i++) {
        ret = ret * 10 + (s[i] - '0');
    }
    return ret;
}
//n位数的最小回文数
//1 + n-2个0 + 1
string smallest(int n) {
    string ret = "";
    for (int i = 0;i < n;i++) {
        if (i == 0 || i == n - 1) {
            ret.push_back('1');
        } else {
            ret.push_back('0');
        }
    }
    return ret;
}
//n位数的最大回文数
//当然是n个9啦
string largest(int n) {
    string ret = "";
    for (int i = 0;i < n;i++) {
        ret.push_back('9');
    }
    return ret;
}
//n本身是个回文数，寻找下一个回文数
string calnext(string n) {
    string half = "";
    bool odd;
    string next = "";
```

```
        if (n.size() & 1) {
            half = n.substr(0, n.size() / 2 + 1);
            odd = true;
        } else {
            half = n.substr(0, n.size() / 2);
            odd = false;
        }
        next = itoa(atoi(half) + 1);
        //如果进位了，就转为n+1位数的最小回文数
        if (next.size() != half.size()) {
            return smallest(n.size() + 1);
        } else {
            return tranToPalindromic(next, odd);
        }
    }
    //n本身是个回文数，寻找上一个回文数
    string calprev(string n) {
        string half = "";
        bool odd;
        string prev = "";
        if (n.size() & 1) {
            half = n.substr(0, n.size() / 2 + 1);
            odd = true;
        } else {
            half = n.substr(0, n.size() / 2);
            odd = false;
        }
        prev = itoa(atoi(half) - 1);
        if (prev.size() != half.size() || n.size() == 2 && prev == "0") {
            return largest(n.size() - 1);
        } else {
            return tranToPalindromic(prev, odd);
        }
    }
    string nearestPalindromic(string n) {
        string res;
        //先把n转成回文数，放弃后一半
        if (n.size() & 1) {
            res = tranToPalindromic(n.substr(0, n.size() / 2 + 1), true);
        } else {
            res = tranToPalindromic(n.substr(0, n.size() / 2), false);
        }
        string next, prev;
        if (res == n) {
            //如果n正好是回文数，那么需要求上一个和下一个来比，因为不能返回本身
            next = calnext(res);
            prev = calprev(res);
        } else if (res < n) {
            //如果res比n小，说明没有必要再去寻找比res小的回文数
            prev = res;
            next = calnext(res);
        } else {
            //如果res比n大，说明没有必要再去寻找比res大的回文数
            next = res;
            prev = calprev(res);
        }
        if (atoi(next) - atoi(n) < atoi(n) - atoi(prev)) {
            return next;
        } else {
            return prev;
        }
    }
};
```

java beats 98.74%

```
class Solution {
    //将字符串倒过来
```

```
private String transpose(String s) {
    StringBuilder ret = new StringBuilder();
    for (int i = s.length() - 1;i >= 0;i--) {
        ret.append(s.charAt(i));
    }
    return ret.toString();
}
//n位开头转成回文数，odd表示是否奇数位
private String tranToPalindromic(String n, boolean odd) {
    if (odd) {
        return n + transpose(n.substring(0, n.length() - 1));
    } else {
        return n + transpose(n);
    }
}
//将数字转成字符串
private String itoa(long n) {
    return String.valueOf(n);
}
//将字符串转成数字
private long atoi(String s) {
    return Long.parseLong(s);
}
//n位数的最小回文数
//1 + n-2个0 + 1
private String smallest(int n) {
    StringBuilder ret = new StringBuilder();
    for (int i = 0;i < n;i++) {
        if (i == 0 || i == n - 1) {
            ret.append('1');
        } else {
            ret.append('0');
        }
    }
    return ret.toString();
}
//n位数的最大回文数
//当然是n个9啦
private String largest(int n) {
    StringBuilder ret = new StringBuilder();
    for (int i = 0;i < n;i++) {
        ret.append('9');
    }
    return ret.toString();
}
//n本身是个回文数，寻找下一个回文数
private String calnext(String n) {
    String half = "";
    boolean odd;
    String next = "";
    if (n.length() % 2 == 1) {
        half = n.substring(0, n.length() / 2 + 1);
        odd = true;
    } else {
        half = n.substring(0, n.length() / 2);
        odd = false;
    }
    next = itoa(atoi(half) + 1);
    //如果进位了，就转为n+1位数的最小回文数
    if (next.length() != half.length()) {
        return smallest(n.length() + 1);
    } else {
        return tranToPalindromic(next, odd);
    }
}
//n本身是个回文数，寻找上一个回文数
private String calprev(String n) {
    String half = "";
    boolean odd;
```

```
        String prev = "";
        if (n.length() % 2 == 1) {
            half = n.substring(0, n.length() / 2 + 1);
            odd = true;
        } else {
            half = n.substring(0, n.length() / 2);
            odd = false;
        }
        prev = itoa(atoi(half) - 1);
        if (prev.length() != half.length() || n.length() == 2 && "0".equals(prev)) {
            return largest(n.length() - 1);
        } else {
            return tranToPalindromic(prev, odd);
        }
    }
    public String nearestPalindromic(String n) {
        String res;
        //先把n转成回文数，放弃后一半
        if (n.length() % 2 == 1) {
            res = tranToPalindromic(n.substring(0, n.length() / 2 + 1), true);
        } else {
            res = tranToPalindromic(n.substring(0, n.length() / 2), false);
        }
        String next, prev;
        if (atoi(res) == atoi(n)) {
            //如果n正好是回文数，那么需要求上一个和下一个来比，因为不能返回本身
            next = calnext(res);
            prev = calprev(res);
        } else if (atoi(res) < atoi(n)) {
            //如果res比n小，说明没有必要再去寻找比res小的回文数
            prev = res;
            next = calnext(res);
        } else {
            //如果res比n大，说明没有必要再去寻找比res大的回文数
            next = res;
            prev = calprev(res);
        }
        if (atoi(next) - atoi(n) < atoi(n) - atoi(prev)) {
            return next;
        } else {
            return prev;
        }
    }
}
```

go

beats 100%

```
//将字符串倒过来
func transpose(s string) string {
 ret := bytes.Buffer{}
 for i := len(s) - 1;i >= 0;i-- {
  ret.WriteByte(s[i])
 }
 return ret.String()
}

//n位开头转成回文数，odd表示是否奇数位
func tranToPalindromic(n string, odd bool) string {
 ret := bytes.Buffer{}
 ret.WriteString(n);
 if odd {
  ret.WriteString(transpose(n[0:len(n) - 1]))
 } else {
  ret.WriteString(transpose(n))
 }
```

```
 return ret.String()
}

//将数字转成字符串
func itoa(n int64) string {
 return strconv.FormatInt(n, 10)
}

//将字符串转成数字
func atoi(s string) int64 {
 ret, _ := strconv.ParseInt(s, 10, 64)
 return ret
}

//n位数的最小回文数
//1 + n-2个0 + 1
func smallest(n int) string {
 ret := bytes.Buffer{}
 ret.WriteRune('1')
 ret.WriteString(strings.Repeat("0", (n - 2)))
 ret.WriteRune('1')
 return ret.String()
}

//n位数的最大回文数
//当然是n个9啦
func largest(n int) string {
 return strings.Repeat("9", n)
}

//n本身是个回文数，寻找下一个回文数
func calnext(n string) string {
 half := ""
 odd := false
 next := ""
 if len(n) % 2 == 1 {
  half = n[0: len(n) / 2 + 1]
  odd = true
 } else {
  half = n[0: len(n) / 2]
  odd = false
 }
 next = itoa(atoi(half) + 1)
 //如果进位了，就转为n+1位数的最小回文数
 if len(next) != len(half) {
  return smallest(len(n) + 1)
 } else {
  return tranToPalindromic(next, odd)
 }
}

//n本身是个回文数，寻找上一个回文数
func calprev(n string) string {
 half := ""
 odd := false
 prev := ""
 if len(n) % 2 == 1 {
  half = n[0: len(n) / 2 + 1]
  odd = true
 } else {
  half = n[0: len(n) / 2]
  odd = false
 }
 prev = itoa(atoi(half) - 1)
 if (len(prev) != len(half) || len(n) == 2 && prev == "0") {
  return largest(len(n) - 1)
 } else {
  return tranToPalindromic(prev, odd)
 }
```

```
}

func nearestPalindromic(n string) string {
    var res string
 //先把n转成回文数，放弃后一半
 if len(n) % 2 == 1 {
  res = tranToPalindromic(n[0: len(n) / 2 + 1], true)
 } else {
  res = tranToPalindromic(n[0: len(n) / 2], false)
 }
 var next, prev string;
 if res == n {
  //如果n正好是回文数，那么需要求上一个和下一个来比，因为不能返回本身
  next = calnext(res)
  prev = calprev(res)
 } else if res < n {
  //如果res比n小，说明没有必要再去寻找比res小的回文数
  prev = res
  next = calnext(res)
 } else {
  //如果res比n大，说明没有必要再去寻找比res大的回文数
  next = res
  prev = calprev(res)
 }
 if (atoi(next) - atoi(n) < atoi(n) - atoi(prev)) {
  return next
 } else {
  return prev
 }
}
```

思路2 时间复杂度: O(len(N)) 空间复杂度: O(1)

后面我觉得完全可以重构一下代码，candidate1 和 candidate2 不用非得算出来，我只要把所有的可能性全都放到一个list里面去，最后来判断就行了

python beats 98.2%

```
class Solution:
    def nearestPalindromic(self, n: str) -> str:
        prefix = int(n[:(len(n)+1)//2])

        candidates = set(['1' + '0' * (len(n)-1) + '1', '9' * (len(n)-1)]) # 进位减位可能性

        for i in map(str, [prefix-1, prefix, prefix+1]): # 前半部分+1，-1，+0可能性
            candidates.add(i + [i, i[:-1]][len(n) & 1][::-1])

        candidates.discard(n) # 除去自身可能就是Palindrome的可能性
        candidates.discard('') # 输入n为个位数的话，我们还会加入空字符串，必须要去掉

        return min(candidates, key = lambda x: (abs(int(x) - int(n)), int(x)))
```

c++ beats 100%

```
class Solution {
public:
    //将字符串倒过来
    string transpose(string s) {
        string ret = "";
        for (int i = s.size() - 1;i >= 0;i--) {
            ret.push_back(s[i]);
        }
        return ret;
    }
```

```
//将数字转成字符串
string itoa(long long n) {
    string ret = "";
    if (n == 0) {
        return "0";
    }
    while (n != 0) {
        ret.push_back('0' + (n % 10));
        n /= 10;
    }
    return transpose(ret);
}
//将字符串转成数字
long long atoi(string s) {
    long long ret = 0;
    for (int i = 0;i < s.size();i++) {
        ret = ret * 10 + (s[i] - '0');
    }
    return ret;
}
//n位数的最小回文数
//1 + n-2个0 + 1
string smallest(int n) {
    string ret = "";
    for (int i = 0;i < n;i++) {
        if (i == 0 || i == n - 1) {
            ret.push_back('1');
        } else {
            ret.push_back('0');
        }
    }
    return ret;
}
//n位数的最大回文数
//当然是n个9啦
string largest(int n) {
    string ret = "";
    for (int i = 0;i < n;i++) {
        ret.push_back('9');
    }
    return ret;
}
long long abs(long long x) {
    return x < 0 ? -x : x;
}
string nearestPalindromic(string n) {
    long long prefix = atoi(n.substr(0, (n.size() + 1) / 2));
    set<string> candidates;
    candidates.insert(smallest(n.size() + 1));
    candidates.insert(largest(n.size() - 1));

    for (auto x : {prefix - 1, prefix, prefix + 1}) {
        string y = itoa(x);
        if (n.size() % 2 == 0) {
            y += transpose(y);
        } else {
            y += transpose(y.substr(0, y.size() - 1));
        }
        candidates.insert(y);
    }
    if (candidates.find(n) != candidates.end()) {
        candidates.erase(candidates.find(n));
    }
    if (candidates.find("") != candidates.end()) {
        candidates.erase(candidates.find(""));
    }
    vector<string> ret(candidates.begin(), candidates.end());

    //排序，跟n距离小的排在前面
```

```
        sort(ret.begin(), ret.end(), [this, n](string x, string y) {
            if (abs(atoi(n) - atoi(x)) != abs(atoi(n) - atoi(y))) {
                return abs(atoi(n) - atoi(x)) < abs(atoi(n) - atoi(y));
            }
            return atoi(x) < atoi(y);
        });

        return ret[0];
    }
};
```

java beats 5%

```
class Solution {
    //将字符串倒过来
    private String transpose(String s) {
        StringBuilder ret = new StringBuilder();
        for (int i = s.length() - 1;i >= 0;i--) {
            ret.append(s.charAt(i));
        }
        return ret.toString();
    }
    //将数字转成字符串
    private String itoa(long n) {
        return String.valueOf(n);
    }
    //将字符串转成数字
    private long atoi(String s) {
        return Long.parseLong(s);
    }
    //n位数的最小回文数
    //1 + n-2个0 + 1
    private String smallest(int n) {
        StringBuilder ret = new StringBuilder();
        for (int i = 0;i < n;i++) {
            if (i == 0 || i == n - 1) {
                ret.append('1');
            } else {
                ret.append('0');
            }
        }
        return ret.toString();
    }
    //n位数的最大回文数
    //当然是n个9啦
    private String largest(int n) {
        StringBuilder ret = new StringBuilder();
        for (int i = 0;i < n;i++) {
            ret.append('9');
        }
        return ret.toString();
    }
    public String nearestPalindromic(String n) {
        long prefix = atoi(n.substring(0, (n.length() + 1) / 2));
        Set<String> candidates = new HashSet<String>();
        candidates.add(smallest(n.length() + 1));
        candidates.add(largest(n.length() - 1));

        for (long x : Arrays.asList(prefix - 1, prefix, prefix + 1)) {
            String y = itoa(x);
            if (n.length() % 2 == 0) {
                y += transpose(y);
            } else {
                y += transpose(y.substring(0, y.length() - 1));
            }
            candidates.add(y);
        }
        candidates.remove(n);
```

```
        candidates.remove(n);
        candidates.remove("");

        final long nLong = atoi(n);
        //先按跟n的距离排序，再按本身排序
        return itoa(candidates
            .stream()
            .map((x) -> atoi(x))
            .sorted(
                (x, y) -> {
                    if (Math.abs(x - nLong) != Math.abs(y - nLong)) {
                        return Long.compare(Math.abs(x - nLong), Math.abs(y - nLong));
                    }
                    return Long.compare(x, y);
                }
            )
            .collect(Collectors.toList())
            .get(0)
        );
    }
}
```

go beats 69%

```
func nearestPalindromic(n string) string {
    prefix, _ := strconv.Atoi(n[:(len(n)+1)/2])
    candidates := map[string]bool{}

    candidates["1" + strings.Repeat("0", len(n)-1) + "1"] = true // 进位减位可能性
    candidates[strings.Repeat("9", len(n)-1)] = true

    var reverse = func(s string) string {
        runes := []rune(s)
        for i, j := 0, len(runes)-1; i < j; i, j = i+1, j-1 {
            runes[i], runes[j] = runes[j], runes[i]
        }
        return string(runes)
    }

    for _, p := range []int{prefix-1, prefix, prefix+1} { // 前半部分+1，-1，+0可能性
        i := strconv.Itoa(p)
        candidates[i + reverse([]string{i, i[:len(i)-1]}[len(n) & 1])] = true
    }

    delete(candidates, n) // 除去自身可能就是Palindrome的可能性
    delete(candidates, "") // 输入n为个位数的话，我们还会加入空字符串，必须要去掉

    keys := make([]int, 0)
    for key := range candidates {
        keyInt , _ := strconv.Atoi(key)
        keys = append(keys, keyInt)
    }

    nn, _ := strconv.Atoi(n)
    byAbs := func(a int, b int) bool {
        if math.Abs(float64(keys[a]) - float64(nn)) != math.Abs(float64(keys[b]) - float64(nn)) {
            return math.Abs(float64(keys[a]) - float64(nn)) < math.Abs(float64(keys[b]) - float64(nn))
        }
        if keys[a] != keys[b] {
            return keys[a] < keys[b]
        }

        return false
    }
    sort.Slice(keys, byAbs)
    res := strconv.Itoa(keys[0])
    keys = make([]int, 0)
    return res
}
```

总结

- 拿到题目如果没有思路怎么办呢？当然是问女朋友了
- 那么如果没有女朋友呢？找呗。。。
- 找不到怎么办？刷好题接着找。。。

}