17 Z 字形变换

更新时间：2019-08-27 09:37:27

> 构成我们学习最大障碍的是已知的东西，而不是未知的东西。
>
> —— 贝尔纳

刷题内容

> 难度：Medium
>
> 原题链接：https://leetcode-cn.com/problems/zigzag-conversion/

内容描述

将一个给定字符串根据给定的行数，以从上往下、从左到右进行 Z 字形排列。

比如输入字符串为 "LEETCODEISHIRING" 行数为 3 时，排列如下：

L C I R
E T O E S I I G
E D H N
之后，你的输出需要从左往右逐行读取，产生出一个新的字符串，比如："LCIRETOESIIGEDHN"。

请你实现这个将字符串进行指定行数变换的函数：

string convert(string s, int numRows);

示例 1:

输入: s = "LEETCODEISHIRING", numRows = 3
输出: "LCIRETOESIIGEDHN"

示例 2:

输入: s = "LEETCODEISHIRING", numRows = 4
输出: "LDREOEIIECIHNTSG"
解释:

L D R
E OE II
EC IH N
T S G

> 题目详解

- 我们首先要考虑到一些特殊情况，比如空字符串的情况，还有numRows 比字符串长度大的情况。

解题方案

思路 1：时间复杂度: O(N) 空间复杂度: O(N)

观察一下每一行的每个字符在原字符串的位置：

- 第一行和最后一行，两个相邻字符之间，在原字符串位置，相差 `(numRows-1)*2`；
- 中间的行，相邻字符之间，在原字符串的位置，都以 `(numRows-1)` 的倍数为中间数成为等差数列，并且差值比 `numRows-1` 小。也就是，假设上一个字符在原字符串的位置是 `j` ，这个字符距离下一个 `numRows-1` 的倍数最近的是，`j + (numRows - 1) - (j % (numRows - 1))`，那么下一个字符在原字符串的位置是，`j + 2 * ((numRows - 1) - (j % (numRows - 1)))`；
- 注意，如果 `numRows为1` 时，`numRows-1为0`，此时上述公式失效，需要特判。

Python beats 33.52%

```python
class Solution:
    def convert(self, s: str, numRows: int) -> str:
        if numRows == 1: # numRows为1时需要特判
            return s
        res, step = '', numRows - 1
        for i in range(numRows):
            j = i
            while j < len(s):
                res += s[j]
                if i == 0 or i == numRows - 1: # 第一行和最后一行
                    j += 2 * step
                else: # 其它行
                    j += 2 * (step - j % step)
        return res
```

Java beats 24.06%

```java
class Solution {
    public String convert(String s, int numRows) {
        if (numRows == 1) { // numRows为1时需要特判
            return s;
        }
        String res = "";
        int step = numRows - 1;
        for (int i = 0; i < numRows; i++) {
            int j = i;
            while (j < s.length()) {
                res += s.charAt(j);
                if (i == 0 || i == numRows - 1) { // 第一行和最后一行
                    j += 2 * step;
                } else { // 其它行
                    j += 2 * (step - j % step);
                }
            }
        }
        return res;
    }
}
```

C++ beats 65.89%

```cpp
class Solution {
public:
    string convert(string s, int numRows) {
        if (numRows == 1) { // numRows为1时需要特判
            return s;
        }
        string ret = "";
        int step = numRows - 1;
        for (int i = 0;i < numRows;i++) {
            int j = i;
            while (j < s.size()) {
                ret.push_back(s[j]);
                if (i == 0 || i == numRows - 1) { // 第一行和最后一行
                    j += step * 2;
                } else { // 其它行
                    j += 2 * (step - j % step);
                }
            }
        }
        return ret;
    }
};
```

go beats 55.82%

```
func convert(s string, numRows int) string {
    if numRows == 1 { // numRows为1时需要特判
        return s
    }
    ret := ""
    step := numRows - 1
    for i := 0;i < numRows;i++ {
        j := i
        for j < len(s) {
            ret += string(s[j])
            if i == 0 || i == numRows - 1 { // 第一行和最后一行
                j += step * 2
            } else { // 其它行
                j += 2 * (step - j % step)
            }
        }
    }
    return ret
}
```

思路 2：时间复杂度：O(N) 空间复杂度：O(N)

刚才的思路一我们需要推公式，公式有的时候还不适用，需要特判，所以我们不如暴力模拟。并且思路1需要取模运算，它的效率很低。

`idx` 从 `0` 开始，自增直到 `numRows-1`，此后又一直自减到 `0`，重复执行。

给个例子容易懂一些：`s = “abcdefghijklmn”`，`numRows = 4`

```
a  g  m
b f h l n
c e i k
d  j
```

从第一行开始往下，走到第四行又往上走，这里用 `step = 1` 代表往下走， `step = -1` 代表往上走。

因为只会有一次遍历，同时把每一行的元素都存下来，所以时间复杂度和空间复杂度都是 `O(N)`。

Python

beats 99.31%

```
class Solution:
    def convert(self, s: str, numRows: int) -> str:
        if numRows == 1 or numRows >= len(s): # 只有一行，或者每一行只有一个元素
            return s

        res = [''] * numRows
        idx, step = 0, 1

        for c in s:
            res[idx] += c
            if idx == 0:  # 第一行，一直向下走
                step = 1
            elif idx == numRows - 1: # 最后一行了，向上走
                step = -1
            idx += step
        return ''.join(res)
```

Java

beats 61.84%

```java
class Solution {
    public String convert(String s, int numRows) {
        // 只有一行，或者每一行只有一个元素
        if (numRows == 1 || s.length() <= numRows) {
            return s;
        }
        StringBuffer[] buffers = new StringBuffer[numRows];
        for (int i = 0; i < buffers.length; i++) {
            buffers[i] = new StringBuffer("");
        }
        int idx = 0;
        int step = 1;
        for (int i = 0; i < s.length(); i++) {
            buffers[idx].append(s.charAt(i));
            // 如果在第一行，就往下走
            if (idx == 0) {
                step = 1;
            }
            // 如果在最后一行，就往上走
            if (idx == numRows - 1) {
                step = -1;
            }
            idx += step;
        }
        String res = "";
        for (int i = 0; i < buffers.length; i++) {
            res += buffers[i];
        }
        return res;
    }
}
```

Go

beats 36.18%

```go
func convert(s string, numRows int) string {
 if numRows == 1 || numRows >= len(s) { // 只有一行，或者每一行只有一个元素
  return s
 }

 idx, step := 0, 1
 lookup := map[int][]byte{}
 for k:= 0; k < numRows; k += 1 {
  lookup[k] = []byte{}
 }
 for i := 0; i < len(s); i += 1 {
  lookup[idx] = append(lookup[idx], s[i])
  if idx == 0 { // 第一行，一直向下走
   step = 1
  } else if idx == numRows - 1 { // 最后一行了，向上走
   step = -1
  }
  idx += step
 }
```

C++

beats 98.92%

```
class Solution {
public:
    string convert(string s, int numRows) {
        if (numRows == 1) {
            return s;
        }
        vector<string> a(numRows, "");
        //step表示方向，1表示向下走，-1表示向上走
        int step = 1;
        //当前x轴位置
        int now = 0;
        for (int i = 0;i < s.size();i++) {
            a[now].push_back(s[i]);
            now += step;
            if (now == numRows - 1) {
                step = -1;
            }
            if (now == 0) {
                step = 1;
            }
        }
        string ret = "";
        for (int i = 0;i < numRows;i++) {
            for (int j = 0;j < a[i].size();j++) {
                ret.push_back(a[i][j]);
            }
        }
        return ret;
    }
};
```

小结

有的时候我们上来就会想到暴力解法，就像这里的思路2，然后我们想着说能不能做一些优化，于是想到一个数学上的办法，觉得自己很厉害，但其实暴力解法可能会更好一些，至少在某些简单场景下。

}

16 最长回文子串

18 字符串转换整数 (atoi)