【第三十一周】哈弗曼编码（Halfman-Coding）与二叉字典树

1、89. 格雷编码

1.

假设已经知道了 n 位格雷编码 $Gray(n)$，通过以下操作，可构造出 $Gray(n+1)$

- n 位格雷码：$Gray(n) = [G_1, G_2, \ldots, G_{2^n-1}, G_{2^n}]$
- 倒序格雷码：$Gray^{-1}(n) = [G_{2^n}, G_{2^n-1}, \ldots, G_2, G_1]$
- 最高位加 1：$Gray'(n) = [2^n + G_{2^n}, 2^n + G_{2^n-1}, \ldots, 2^n + G_2, 2^n + G_1]$
- 拼接集合：$Gray(n+1) = Gray(n) + Gray'(n)$

通过以上方法，可从 0 位开始推导任意位数格雷编码。

```
 /**
 * @param {number} n
 * @return {number[]}
 */
//  构造法
var grayCode = function(n) {
    let len = 1 << n;
    let ret = new Array(len);
    if (n == 0) { //边界条件是0阶
    //末尾是阶0
        ret[0] = 0;
        return ret;
    }
    // 先获得 n - 1阶的格雷码
    let code_n_1 = grayCode(n - 1);
    let len_n_1 = code_n_1.length
    // 用 构造n阶的格雷码
    for (let i = 0; i < len_n_1; i++) {
        ret[i] = code_n_1[i] << 1;
        //  ret[i]  对称的位置是 ret[2 * len_n_1 - i - 1]
        // 末尾阶1
        ret[2 * len_n_1 - i - 1] = code_n_1[i] << 1 | 1;
    }
    return ret;
};
```

2、面试题 17.17. 多次搜索

1. 通过smalls构建Trie树，遍历big，在前缀树中寻找匹配项。
2. 找到完整字符则将当前的「索引」push进结果集，last为结果集的index。
3. 继续向下找，直到匹配不到/big循环完毕

4. 如果未能找到则break，进入下一个字符的循环。

```
 function TreeNode(val) {
    this.val = val || null
    this.children = {}
}

/**
 * @param {string} big
 * @param {string[]} smalls
 * @return {number[][]}
 */
var multiSearch = function (big, smalls) {
    const res = smalls.map(() => [])
    if (!big) {
        return res
    }
    let Tree = new TreeNode()
    let now;
    smalls.forEach((small, index) => {
        now = Tree;
        for (let i = 0; i < small.length; i++) {
            if (!now.children[small[i]]) {
                now.children[small[i]] = new TreeNode(small[i])
            }
            now = now.children[small[i]]
        }
        now.children['last'] = index
    })

    for (let i = 0; i < big.length; i++) {
        let now = Tree;
        for (let j = i; j < big.length; j++) {
            if (!now.children[big[j]]) {
                break
            }
            now = now.children[big[j]]
            if (now.children.last !== undefined) {
                res[now.children.last].push(i)
            }
        }
    }
    return res
};
```

3、468. 验证IP地址

1. IPv6地址比较好判断：直接判断8组数均为4位以内16进制数即可/^[0-9a-fA-F]{1,4}$/
2. IPv4的话，如果用正则表达式判断每组数小于256比较繁杂。
3. 这里先用正则判断是否为3位数字以内/^0$|^[1-9]\d{0,2}$/（注意单个0要单独判断，避免出现01.01.01.01这样的情况），再判断数字是否小于256即可。

```
 /**
```

```
 * @param {string} queryIP
 * @return {string}
 */
var validIPAddress = function(IP) {
    const arr4 = IP.split(".");
    const arr6 = IP.split(":");
    if (arr4.length === 4) {
        if (arr4.every(part => (part.match(/^0$|^([1-9]\d{0,2})$/) && part <
256) )) {
            return "IPv4";
        }
    } else if (arr6.length === 8) {
        if (arr6.every(part => part.match(/^[0-9a-fA-F]{1,4}$/))) {
            return "IPv6";
        }
    }
    return "Neither";
};
```

4、32. 最长有效括号

1. 状态值：dp[i] 状：s中以 ')' 结尾的最长有效括号的长度
2. 状态转移方程：也就是找dp[i]和dp[i - 1]之间关系状态

```
/**
 * @param {string} s
 * @return {number}
 */
var longestValidParentheses = function(s) {
    let n = s.length;
    let dp = Array(n).fill(0);
    for (let i = 0; i < n; i ++) {
        if (s[i] === ')') { //只考虑i以‘)’结尾，否则不可能是有效括号
            let k = dp[i - 1];
            //要想和i-1扯上关系，那么就要看i-1前面那个，也就是 i - k - 1 为下标的s是否存在
且对应的是不是'('
            if (i - k - 1 >= 0 && s[i - k - 1] === '('){
                dp[i] = dp[i - 1] + 2; //是的话+2
                //如果前面还存在有效的括号对，直接加进来
                if (i - k - 2 > 0) {
                    dp[i] += dp[i - k - 2];
                }
            }
        }
    }
    return Math.max(...dp, 0)
};
```