【第三十六周】有趣的莫比乌斯反演

1、1447. 最简分数

1. 分子分母大小都在[1,n]区间内。
2. 分子为1时，分母除了1其他都有效。
3. 公约数最大出现在"分子/2"的位置。

```
/**
 * @param {number} n
 * @return {string[]}
 */
var simplifiedFractions = function(n) {
    let result = []
    // i记录分子
    for(let i=1;i<=n;i++){
        // j记录分母
        out: for(let j=2;j<=n;j++){
            // 分子为1必成立
            // 分子必须大于分母
            // 增加i % j !== 0条件是为了减少公约数判断的循环次数
            if(i === 1 || (j > i && j % i !== 0)){
                // 公约数校验
                for(let o=2;o<=i/2;o++){
                    if(j % o === 0 && i % o === 0){
                        continue out
                    }
                }
                result.push(`${i}/${j}`)
            }
        }
    }
    return result
};
```

2、878. 第 N 个神奇数字

1. 小于等于 x 的神奇数字的个数是一个单调递增函数，可以用二分搜索来做这道题。

设 $L = lcm(A, B)$，为 A，B 的 **最小公倍数**，$L = \frac{A*B}{gcd(A,B)}$。

2. $f(x)$ 为小于等于 x 的神奇数字的个数。$f(x) = \lfloor \frac{x}{A} \rfloor + \lfloor \frac{x}{B} \rfloor - \lfloor \frac{x}{L} \rfloor$。有 $\lfloor \frac{x}{A} \rfloor$ 个数字能被 A 整除的，$\lfloor \frac{x}{B} \rfloor$ 个数字能被 B 整除，同时需要减去 $\lfloor \frac{x}{L} \rfloor$ 个能被 A，B 整除的数。

```
/**
 * @param {number} n
 * @param {number} a
 * @param {number} b
 * @return {number}
 */
var nthMagicalNumber = function(N, A, B) {
    gcd = (x, y) => {
        if (x == 0) return y;
        return gcd(y % x, x);
    }

    const MOD = 1000000007;
    const L = A / gcd(A, B) * B;

    let lo = 0;
    let hi = 1e15;
    while (lo < hi) {
        let mi = lo + Math.trunc((hi - lo) / 2);
        if (Math.trunc(mi/A) + Math.trunc(mi/B) - Math.trunc(mi/L) < N)
            lo = mi + 1;
        else
            hi = mi;
    }

    return lo % MOD;
};
```

3、372. 超级次方

1. 快速幂的思想，例如，求解a^100是通过(a^10)^10进行求解，而不是对a进行100次的累乘。主要参考幂的乘方
2. 幂的乘方：$(p^a)^b = p^{a*b}$

```
/**
* @param {number} a
```

```
 * @param {number[]} b
 * @return {number}
 */
var superPow = function(a, b) {
    const MOD = 1337;
    let res = 1;
    a %= MOD;
    for(let i = b.length - 1; i >= 0; i--){
        let k = 1;
        for(let j = 0; j < 10; j++){
            if(j == b[i]) res = res * k % MOD;
            k = k * a % MOD;
        }
        a = k;
    }
    return res;
};
```

4、60. 排列序列

1. 利用 hashSet 存储选过的数字，避免重复选择（剪支）。
2. 当选齐了，就生成了一个排列，用变量记录它是第几个，等于 k 就返回它。
3. 不等于 k，就结束当前递归，回溯，撤销最后一个选择，进入别的分支继续搜索。
4. 一旦找到了第 k 个排列，就没必要往右搜，想办法控制一下，我选择根据递归函数返回值判断，在 for 循环中 return。
5. 全排列问题很自然想到了回溯

```
/**
 * @param {number} n
 * @param {number} k
 * @return {string}
 */
var getPermutation = function(n, k) {
  const used = new Set();

  let groupNum = 1;
  for (let i = 1; i <= n; i++) {
    groupNum = groupNum * i;
  }

  const helper = (temp) => {       // temp是当前已选的数字数组
    const progress = temp.length; // progress表示当前已选了几个数字

    if (progress == n) {           // 因为是空降到正确的组，选够了n个即可返回
      return temp.join('');
```

```
    }

    groupNum = groupNum / (n - progress); // 一个分组有多少个

    for (let i = 1; i <= n; i++) {
      if (used.has(i)) continue;

      if (k > groupNum) { // k大于一组的个数
        k = k - groupNum; // 更新k,
        continue;         // 跳过这一组，即跳过当前的数字i
      }
      temp.push(i);         // 选择i
      used.add(i);          // 记录选择
      return helper(temp); // 进度+1 继续选
    }
  };

  return helper([]);
};
```

5、1512. 好数对的数目

1. 所谓的好数字，实际就是统计重复数字的次数，因此直接用map来记录。key：数字，value：重复次数。这样一次循环，就可以统计完成。

```
/**
 * @param {number[]} nums
 * @return {number}
 */
var numIdenticalPairs = function(nums) {

    let map = {};
    let pairs = 0;
    for(let i = 0; i < nums.length; i++) {
        let key = String(nums[i]);
        map[key] = map[key] === undefined ? 0 : map[key] + 1;
        pairs += map[key];
    }

    return pairs;
};
```

6、1359. 有效的快递序列数目

1. 大部分动态规划的题目本质上可以归结为组合数学的问题。
2. 当n = 1时，唯一可行排列是(P1, D1),
3. 当n = 2时，P2可以放在三个槽位：
4. P1的前面，P1和D1的中意，D1的后面。
5. 在这三种情况下D2的位置分别有3、2、1个选择，总共6种情形。如下所示：

   (P2,D2,P1,D1)，(P2,P1,D2,D1)，(P2,P1,D1,D2)

   (P1,P2,D2,D1)，(P1,P2,D1,D2)

   (P1,D1,P2,D2)
6. 可以发现，有效序列的数量是槽位数x的累加和，用高斯算法即可快速求解。
7. 对于n > 1的情况，槽的个数为i*2-1，累加求和以后和n-1的数量相乘即可得到结果。

```
/**
 * @param {number} n
 * @return {number}
 */
var countOrders = function (n) {
    const mod = 1e9 + 7;
    let count = 1;
    for (let i = 1; i < n; i++) {
        let x = i * 2 + 1;
        count *= (1 + x) * x / 2;
        count %= mod;
    }
    return count;
};
```