279. 完全平方数

思路1:

动态规划，对于任何一个数,如果是一个平方数,那么,平方根就是求解的值,结果值为1

1. 初始值: dp[1] = 1;
2. j : [2,sqrt]
3. 单个元素的初始值: dp[i] = i; 完全由1组成
4. 开始探测: dp[i] = dp[i-j*j] + 1;
5. 取第四部产生的最小值作为结果值

思路2:

贪心算法

思路3:

暴力求解

263. 丑数

264. 丑数 II

313. 超级丑数

概念:

丑数: 由 2,3,5 作为仅有的质因子的正整数

超级丑数: 指定仅有的质因子列表

思路:

最小数生成器; 因为指定了因子,所以, 该数只能有指定的因子相乘; 那剩下的事情就是，如何生成下一个数了;

以丑数为例: 2(x)3(y)5(z); x == y == z == 0时,是起始值为1;

seed(种子): [2,3,5]

当前最小值: 2; 移出2, 下一个数为 dp中的值 * 2

代码模板:

```
1     public int nthSuperUglyNumber(int n, int[] primes) {
2
3         // 初始化 DP
4         int[] dp = new int[n];
5         dp[0] = 1;
6
7         // 初始化指针
```

```
        int[] ptr = new int[primes.length];
        // 初始化当前值
         int[] values = new int[primes.length];
         System.arraycopy(primes,0,values,0,primes.length);

         // 开始计算能够产生的最小值
         for(int i = 1; i < n; i++){

             // 计算当前值
             int min = Integer.MAX_VALUE;
             for (int value : values) {
                 min = Math.min(min, value);
             }
             dp[i] = min;

             // 当前值已经入选,需要将当前值更新为下一个可选项
             // 防止values中的重复数字对结果产生干扰
             for(int j = 0; j < values.length; j++){
                 if (values[j] == min){
                     // 产生下一个可选项
                     ptr[j]++;
                     values[j] = dp[ptr[j]] * primes[j];
                 }
             }
         }
         return dp[n-1];

    }
```

204. 计数质数

思路1:

顺序检测该值是不是质数; 质数的判断条件为: [2,sqrt(n)] 是不是该值的因子;

思路2: 标记

1. 2为质数,所以2的倍数(不含2)不可能为质数
2. 3为质数,所以3的倍数(不含3)不可能为质数

3. 数学归纳法: 如果某个数为质数,则该数的倍数(不包含该数本身)一定不是质数
4. 标记后: 下一个为标记的数字,就是质数, 因为不能为[2,n-1]数整除;
5. 寻找到sqrt(n); 因为如果一个数的因为如果 > sqrt(n), 那么另一个因为一定会 < sqrt(n)

```
    public int countPrimes(int n) {

        // DP
        boolean[] dp = new boolean[Math.max(2,n)];
        dp[0] = true;
        dp[1] = true;
        // 边界值
        int sqrt = ((Double)(Math.sqrt(n))).intValue();

         // 选取起始的素数
         int nextPrime = 2;

         while(nextPrime <= sqrt){
             // 将素数的倍数作为
             int next = nextPrime << 1;
             while(next < n){
                 dp[next] = true;
                 next += nextPrime;
             }

             // 选取下一个素数
             nextPrime++;
             while(dp[nextPrime] != false){
                 // choose == current
                 nextPrime++;
             }
         }

         // 扫描结果
         int size = 0;
         for(boolean scan : dp){
             if(!scan){
                 size++;
             }
         }
```

```
36
37          return size;
38      }
```

思路3:

标记思路优化; 根据标记思路中可以发现: 偶数是可以忽略的，也就是简单的不考虑偶数的情况, 则可以将原有的操作缩小为 1/2;

```
1   public int countPrimes(int n) {
2
3       if(n <= 2){
4           return 0;
5       }
6
7       // DP
8       boolean[] dp = new boolean[n/2];
9       // 边界值
10       int sqrt = ((Double)(Math.sqrt(n))).intValue();
11
12       // 选取起始的素数
13       int nextIndex = 1;
14       int nextPrime = (nextIndex<<1)+1;
15       while(nextPrime <= sqrt){
16           // System.out.println(nextPrime);
17           // 将素数的倍数作为
18           // next += next << 1;
19           int next = (nextPrime << 1) + nextPrime;
20           while(next < n){
21               dp[next>>1] = true;
22               next += nextPrime << 1;
23           }
24
25           // 选取下一个素数
26           while(++nextIndex < dp.length && dp[nextIndex]){
27           }
28           nextPrime = (nextIndex << 1) +1;
29       }
30
31       // 扫描结果
```

```
        int size = 1;
        for(int i = 1; i < dp.length; i++){
            if(!dp[i]){
                size++;
            }
        }
        return size;
    }
```