素数判断及矩形交集

本节，我们看看两道关于基础数据类型的面试算法题，这两道题在面试中屡次出现，他们难度不大，但要在 45 分钟内处理得当，却也不是容易的事情。题目一是素数判断，给定一个正整数 n,返回 1 到 n 间的素数。题目二是，给定两个二维平面上的矩形，判断矩形是否相交，相交的话返回他们的交集。

我们先看第一题，要返回 1 到 n 间的素数，做法有多种，不同的方法，他们理论上的时间复杂度都差不多，但实践效果却不相同，有些方法在实际运用中，耗时很长，有些方法则需要消耗内存空间，因此候选人在设计算法时，对相关限制条件的考虑和取舍，能很好的反映候选人的理论水平和工程能力。

首先我们考虑最简单的做法，暴力枚举法,从 1 到 n 进行遍历，对处于 1 到 n 中的每一个数 k,判断 k 是否是素数，如果 k 是素数，则将 k 加入一个素数集合。因此，接下来的重点是如何判断 k 是一个素数。根据定义，素数只能被 1 和它本身整除，所以，判断 k 是否是素数，只要看每一个比 k 小的数是否能整除 k 即可，由此，判断 k 是否是素数，我们有下

面的伪码:

```
boolean  isPrime(int k) {
    for (int i = 2; i < k; i++) {
    if (k % i == 0)     return false;
    }

    return true;
}
```

于是我们可得到一个可行的算法:

```
Array<int>  getPrimes(int n ) {
    Array<int> primes = new Array<int>();
    for (int i = 1; i <= n; i++) {
        if (isPrime(i)) {
            primes.add(i);
        }
    }

    return primes;
}
```

上面的做法可行，但却有很大的改进空间。改进点在于对一

个数是否是素数的判断，如果一个数不是素数的话，那它能分解成一系列小于它的素数的乘积,例如 8 分解成 2*2*2, 26 分解成 2*13.由此，要判断一个数是否是素数，只要看小于它的素数是否能整除它即可。这样的改进，时间效率的提示是巨大的，特别是随着 k 越来越大，效果就越明显。因为小于 k 的素数的个数要远远小于 k，例如小于 10 的素数有 1，2，3，5，7，是 5 个，而小于 50 的素数只有 18 个，因此我们可以把上面的判断素数的算法进行改进:

```
Array<int> prime_arr = new Array<int>(1, 2, 3);
Boolean isPrime(int k ) {
    if (k <= 3) {
        return true;
    }

    for (int i = 0; i < prime_arr.length(); i++) {
        if (k % prime_arr.get(i) == 0) {
            return false;
        }
    }

    prime_arr.add(k);
```

```
    return true;
}
```

上面的方法还能不能再改进呢，答案是肯定的，上面的算法是，对任意整数，只要该整数不能被小于他的素数整除，那么他就是素数。例如 9 能被 3 整除，所以 9 不是素数。如果我们反其道而行，先拿到一个素数，然后把该素数的倍数对应的数都删除，例如我们先取到素数 2，然后删除所有 2 的倍数，接着取第二个素数 3，删除所有 3 的倍数，反复进行，直到没有可删除的数为止，代码如下：

```
boolean[] primes = new boolean[n+1];
for (int i = 0; i <= n; i++) {
    primes[i] = true;
}

for (int i = 2; i <= n; i++) {
    if (primes[i] == true) {
        int p = i;
        for (int j = 2; j * p <= n; j++) {
            primes[j*p] = false;
        }
```

```
    }
}
```

执行上面代码后，对任意一个处于 1 到 n 中的数 k,如果 primes[k] == true, 那么 k 就是一个素数。在代码的第二个 for 循环里，p 的取值会是 2，3，5，7，11…也就是 p 的取值是依次递增的素数。

第三种方法，其效率又是对第二种的改进，第二种方法，当删除一个非素数时，要遍历所有比该数小的素数，然后依次做除法运算，而第三种方法无需做这种循环遍历，因此效率得到了提高。方法三思路巧妙，独居匠心，能打动面试官的显然是第三种，唯一的缺陷就是需要消耗内存空间。

我们接着看第二题

在二维平面中，如果一个矩形，它的长与 x 坐标轴平行，高与 y 坐标轴平行，那么我们就称矩形是坐标轴对齐的，这样的矩形，在数据结构上，我们用它的左下角坐标(x, y) 以及宽 w 和高 h 来表示。给定两个坐标轴对齐的矩形，判断他们是否相交，如果相交，给出他们相交所形成的矩形。

我们先根据题目设计矩形的数据结构:

```
class Rectangle {
public int x;
public int y;
public int w;
public int h;
};
```

假设两个矩形为 R 和 S, 根据小样本实例检测法，大家可以先画两个相交的矩形，看看他们的点和边有什么关联，然后再画两个不相交的矩形，看看他们的点和边有什么联系。通过若干次尝试，我们可以发现两个矩形 R = ((Rx, Ry), Rw, Rh) , S = ((Sx, Sy), Sw, Sh), 如果满足条件： [Rx, Rx + Rw] 与 [Sx, Sx + Sw] 的交集是空集时，两个矩形不相交，同理，如果[Ry, Ry + Rh] 与 [Sy , Sy + Sh] 的交集是空集时，两个矩形也不相交，因此只有上面两种情况下的集合都相交的话，两个矩形才可能相交：

```
is_interset(Rectangle R, Rectangle S) {
  return R.x <= S.x + S.w && S.x <= R.x + R.w && R.y
<= S.y + S.h &&
   S.y <= R.y + R.h;
}
```

如果两个矩形相交，我们尝试着画出两个相交的矩形，观察一下，不难发现，他们交集部分形成的矩形，左下角的 x 坐标为： max(R.x, S.x)，左下角的 y 坐标为 max(R.y, S.y). 矩形的宽是 min(R.x+R.w, S.x + S.w) - max(R.x, S.x)，高为:min(R.y+R.h , S.y + S.h) - max(R.y, S.y)

这道题难度不大，但是由于其频频出现在面试中，因此特别值得拿出来好好讨论。