链滴

随机洗牌的算法

作者：valarchie

原文链接：https://ld246.com/article/1568726128589

来源网站：链滴

今天偶然看到群里的萌新说道，面试被问如何将扑克牌随机洗牌输出。笔者觉得这道题挺有意思而且开放性，有多种不同的实现方式。然后我就随手写了一个算法，仔细一想这个算法的优化空间挺大，是又写出三种算法。

第一种

我们通过JDK的随机算法获取一个随机下标，再通过Set集合来判断牌是否有被抽到过，如果抽到过的，继续进行循环，直到抽到原牌数量为止。

```
public class ShuffleCard1 {

    public static int[] getShuffleCards(int[] cards) {
        // 获取随机数种子
        Random rand = new Random(System.currentTimeMillis());
        // 用Set集合存储已抽过的牌
        Set<Integer> isExisted = new HashSet();
        // 声明洗牌后数组
        int[] shuffleCards = new int[cards.length];
        // 已抽到的牌数量
        int drawCount = 0;
        // 当抽到的牌数量没达到原牌数组的大小时循环
        while (drawCount < cards.length) {
            // 获取一个随机下标
            int index = rand.nextInt(cards.length);
            // 判断该下标对应的牌是否已被抽过，没有的话，抽出
            if (!isExisted.contains(cards[index])) {
                shuffleCards[drawCount++] = cards[index];
                isExisted.add(cards[index]);
            }
        }
        return shuffleCards;
    }
}
```

第二种

我们分析一下，判断牌是否被抽到的方法可以进一步优化，我们可以使用位数组来进行判断效率更高于是我们将Set改为byte数组判断牌是否抽到。

```
public class ShuffleCard2 {

    public static int[] getShuffleCards(int[] cards) {
        // 获取随机数种子
        Random rand = new Random(System.currentTimeMillis());
        // 利用byte数组来判断该牌是否有被抽到过
        byte[] isExisted = new byte[cards.length];
        // 声明洗牌后数组
        int[] shuffleCards = new int[cards.length];
        // 已抽到的牌数量
        int drawCount = 0;
        // 当抽到的牌数量没达到原牌数组的大小时循环
        while (drawCount < cards.length) {
            // 获取一个随机下标
```

```
            int index = rand.nextInt(cards.length);
            // 如果byte数组对应下标为0的话，代表还未抽到
            if (isExisted[index] == 0) {
                shuffleCards[drawCount++] = cards[index];
                isExisted[index] = 1;
            }

        }

        return shuffleCards;
    }
}
```

第三种

我们分析一下，假设牌组内有54张牌。我们第一次抽到一张牌后，第二次又从原来的数组随机抽取，此时牌已经剩53张牌，但是我们还是从54张牌中进行抽取，所以我们可以提升这部分的效率。于是我在每次抽取牌的时候都缩小抽牌的范围。并且每抽到一张牌，就依次与数组尾部的元素进行交换。假设a,b,c,d,e]五张牌，第一次抽到c，那么c已经被抽到了，就将c移到数组末尾，变为[a,b,d,e,c]。第二次取元素的时候我们就从下标0~3的位置随机抽取，排除掉c元素。依次类推。

```
public class ShuffleCard3 {

    public static int[] getShuffleCards(int[] cards) {
        // 获取随机数种子
        Random rand = new Random(System.currentTimeMillis());
         // 声明洗牌后数组
        int[] shuffleCards = new int[cards.length];
        // 已抽到的牌数量
        int drawCount = 0;
        // 我们通过减少抽牌的范围，从例如先从54开始取随机数，
        // 然后是53依次类推到1。
        for (int i = shuffleCards.length; i > 0; i--) {
            // 获取一个随机下标
            int index = rand.nextInt(i);
            // 填入洗牌后数组
            shuffleCards[drawCount++] = cards[index];
            // 该牌如果已被抽到过，每次都放在数组尾部
            cards[index] = cards[i-1];

        }
        return shuffleCards;
    }
}
```

第四种

由于第一种和第二种算法基本上随机抽取的次数都会大于牌组的数量，因为随机大概率会出现重复。以我们可以转变一下思路，不通过抽取牌来达到洗牌效果，而通过随机交换原数组内的元素来达到随洗牌的目的。这样一来就可以降低随机的次数。

```
public class ShuffleCard4 {

    public static int[] getShuffleCards(int[] cards) {
        // 获取随机数种子
        Random rand = new Random(System.currentTimeMillis());
        // 遍历原牌组
        for (int i  = 0; i < cards.length; i++) {
            // 获取一个随机下标并与之交换
            int index = rand.nextInt(cards.length);

            int tmp = cards[i];
            cards[i] = cards[index];
            cards[index] = tmp;
        }
        return cards;
    }


}
```

测试四种算法

```
 public static void main(String[] args) {

        int[] cards = new int[54];

        for (int i = 0; i < cards.length; i++) {
            cards[i] = i + 1;
        }

        long t1 = System.currentTimeMillis();
        for (int i = 0; i < 1000000; i++) {
            ShuffleCard1.getShuffleCards(cards);
        }
        long t2 = System.currentTimeMillis();
        System.out.println("第一种方法用时：" + (t2 - t1));


        long t3 = System.currentTimeMillis();
        for (int i = 0; i < 1000000; i++) {
            ShuffleCard2.getShuffleCards(cards);
        }
        long t4 = System.currentTimeMillis();
        System.out.println("第二种方法用时：" + (t4 - t3));


        long t5 = System.currentTimeMillis();
        for (int i = 0; i < 1000000; i++) {
            ShuffleCard3.getShuffleCards(cards);
        }
        long t6 = System.currentTimeMillis();
        System.out.println("第三种方法用时：" + (t6 - t5));
```

```
        long t7 = System.currentTimeMillis();
        for (int i = 0; i < 1000000; i++) {
            ShuffleCard4.getShuffleCards(cards);
        }
        long t8 = System.currentTimeMillis();
        System.out.println("第四种方法用时：" + (t8 - t7));


    }
```

测试结果：

```
第一种方法用时：3300ms
第二种方法用时：2214ms
第三种方法用时：572ms
第四种方法用时：543ms
```

如果错误，恳请网友评论指正。