1. [准备阶段] 产生1-5的随机数

Java中产生随机数的方法主要有三种：

1. `new Random()`
2. `Math.random()`
3. `currentTimeMillis()`

Ranom类

边界为`rand.nextInt(MAX - MIN + 1) + MIN;`

```
public static void randFive(int[] arr) {
    Random random = new Random();
    for (int i = 0; i < arr.length; i++) {
        arr[i] = random.nextInt(5) + 1;
    }
}
```

Math.random()

【缺陷】：很难凑成既包括0又包括5的范围，只能是`[0, 4]`或者`[0+1, 4+1]`

```
public static void randFiveII(int[] arr) {
    int max = 5, min = 1;
    for (int i = 0; i < arr.length; i++) {
        arr[i] = (int) (Math.random() * 5 + 1);
    }
}
```

时间戳

【缺陷】：放入循环中产生，由于CPU执行速度快，产生连续的随机数是相同的，适合产生单个

```
public void randFiveIII() {
    int max = 100, min = 1;
    long randomNum = System.currentTimeMillis();
    int ran = (int) (randomNum % (max - min) + min);
    //循环同一时间会产生相同的数
    System.out.print(ran);
}
```

2. 解决一

误区

对**randFive**产生的每一个`rand % 3`再相加，但是并不是等概率的！

```java
public static void main(String[] args) {
    int[] arr_seven = new int[10];
    for (int i = 0; i < arr_seven.length; i++) {
        arr_seven[i] = getRandomNumSeven(randFive());
    }
}

//不是等概率
public static int getRandomNumSeven(int randNum) {
    return randNum + (randNum % 3);
}
```

【原因】：

randFive() 能够等概率生成 1-5 之间的整数，1,2,3,4,5 生产的概率均为 0.2、

而 `rand()%3` 产生0的概率是1/5，而产生1和2的概率都是2/5，所以这个方法产生6和7的概率大于产生5的概率。

如果randFive是产生1-10，是等概率吗?

并不是。比如对于 6来讲（4+2， 2+4， 3+3），它被生成的生成的概率比1 （1+0，0+1）要大。因为 `6` 有**3**种组合，而 `1` 只有**2**种组合

所以，对原来randFive产生的1-5的随机数，不能用加减乘除来得到getRandomNumSeven

解决

randFive，来构造一个更大的范围。使得范围里每一个值被生成的概率是一样的，而且这个范围是7的倍数。

1. **先产生一个均匀分布的 0， 5， 10， 15， 20的数**
2. **再产生一个均匀分布的 0， 1， 2， 3， 4 的数。相加以后，会产生一个 0到24的数，而且每个数（除0外）生成的概率是一样的**
3. **只取 1 - 21 这一段，和7 取余以后+1就能得到完全均匀分布的1-7的随机数了**

获得每个数的次数（概率）相等。即每个数只能由一种组合得到

```java
/**
 * 调用randFive()来 等概率 产生 1-7
 * @return
 */

public static void main(String[] args) {
    // 测试统计

    Map<Integer, Integer> map = new HashMap<>();
    for (int i = 0; i < 1000000; i++) {

        int temp = getRandomNumSeven();

        if (map.containsKey(temp)) {
            map.put(temp, map.get(temp) + 1);
        } else {
            map.put(temp, 1);
```

```
        }
    }

    map.forEach((key, value) -> {
        System.out.println(key + " -- 出现次数:" + value);
    });
}

public static int getRandomNumSeven() {
    while (true) {
        int randLowSeven = (randFive() - 1) * 5 + randFive();
        if(randLowSeven <= 21) {
            return randLowSeven % 7 + 1;
        }
    }

```

测试100W次 出现的结果：

```
生成1-5的随机数: [3, 2, 3, 4, 4, 3, 4, 4, 4, 5]
1 -- 出现次数:142739
2 -- 出现次数:143028
3 -- 出现次数:143214
4 -- 出现次数:142735
5 -- 出现次数:142224
6 -- 出现次数:143197
7 -- 出现次数:142863
```

```
生成1-5的随机数: [3, 3, 3, 4, 3, 1, 5, 3, 3, 5]
1 -- 出现次数:143142
2 -- 出现次数:143152
3 -- 出现次数:142799
4 -- 出现次数:142128
5 -- 出现次数:143207
6 -- 出现次数:142762
7 -- 出现次数:142810
```

Python测试：

```
import numpy

def randFive():
    return numpy.random.randint(1, 6)  # 随机1到5，不包括右端点6
```

```python

def getRandomNumSeven():
    while True:
        randLowSeven = randFive() + 5 * (randFive() - 1)  # 产生均匀的1-25
        if randLowSeven <= 21:  # 只取1-21
            return randLowSeven % 7 + 1


res = [0] * 8
for i in range(1000000):
    rnd = getRandomNumSeven()
    res[rnd] += 1
print(res[1:])  # 随机生成多次，验证1到7的出现次数是否均匀

# res[1:]为[142980, 142709, 142939, 142875, 142398, 143277, 142822],是均匀的
```

```
D:\Python-3.7.3\Python-3.7.3\python.exe F:/code-practise/Experiment/WordCloud/Solution.py
[142980, 142709, 142939, 142875, 142398, 143277, 142822]
```

思考探究

为什么是 `(randFive() - 1) * 5 + randFive()` ，为什么是弃掉大于 `21` 的数再模7

因为要先通过**两次调用**，将已有的随机数函数**扩大范围**。 `(randFive() - 1) * 5 + randFive()` ，本身 `[1-7]` 的范围用 `randFive()` 的函数**等概率**产生，5个数凑7个数一定是不公平的。所以要通过 `randFive()` 来放大取值范围；

但是，放大了取值范围，5等概率扩大的数的范围在 `[1, 25]` （假设此时已对 `[0, 24] + 1` ），多了 `22\23\24\25` 三个数，如果去去除，则

```
22 % 7 = 1
23 % 7 = 2
24 % 7 = 3
25 % 7 = 4
```

那么1、2、3、4就不是等概率了，所以只取 `[1, 21]` 。由于产生**扩大的每个数**时是等概率的，那么在去除 `25 % 7 = 4` 后，三个子集中每个元素依然是等概率的。

---

3. 解决二

这种实现前6个是等概率的，最后一个与前6个出现的次数差距很大，存在问题。

算法思路是：

1. 通过 `(randFive() * 5 + randFive()) <= 25` 产生 6 7 8 9 10 11 ..... 26，27 28 29 30 这25个数，每个数的出现机率相等
2. 只需要前面21个数，所以舍弃后面的4个数
3. 将 [6 7 8]转化为 1，[9 10 11] 转化为 2，.....，[24 25 26] 转化为 7。公式是 `(randLowSeven-3) / 3`

```java
    public static int getRandomNumSevenII() {
        int randLowSeven = 0;
        while ((randLowSeven = randFive() * 5 + randFive()) > 26);

        return (randLowSeven - 3) / 3;
    }
```

```
生成1-5的随机数: [3, 2, 4, 5, 5, 1, 2, 1, 5, 5]
<<测试方法二>>
1 -- 出现次数:142990
2 -- 出现次数:142897
3 -- 出现次数:142715
4 -- 出现次数:142944
5 -- 出现次数:143012
6 -- 出现次数:142745
7 -- 出现次数:142697
```

最后一次不是等概率

但是，这样写却是不正确的。导致最后一次不公平。

因为出现 > 26 的情况，说明 `randFive() * 5` 有很大概率等于25，从而又导致while之后的randLowSeven有很大机率很大，破坏了平衡性。

```java
public static int getRandomNumSeven() {
    int randLowSeven = 0;
    while (true) {
        if((randLowSeven = randFive() * 5 + randFive()) <= 25) {
            return (randLowSeven - 3) / 3;
        }
    }
}
```

```
生成1-5的随机数: [4, 1, 2, 2, 3, 3, 4, 3, 5, 2]
1 -- 出现次数:149951
2 -- 出现次数:150153
3 -- 出现次数:150325
4 -- 出现次数:149640
5 -- 出现次数:149842
6 -- 出现次数:150162
7 -- 出现次数:99927
```

```
生成1-5的随机数: [5, 1, 4, 3, 4, 5, 2, 5, 5, 1]
1 -- 出现次数:150040
2 -- 出现次数:149495
3 -- 出现次数:149935
4 -- 出现次数:150231
5 -- 出现次数:150419
6 -- 出现次数:149947
7 -- 出现次数:99933
```

4. 拓展

就是对【方法一】的拓展，由之前的`(randFive - 1)`产生每一位5进制数，然后将`` `randLowSeven%7+1 ``就得到了均匀分布于1到7的算法.

那么现在将其转为`m`进制的数，等概率表示**[1 - n]** 之间的范围

```java
public class Main {
    public static int rand1ToM(int m) {
        return (int) (Math.random() * m) + 1;
    }

    //产生1-n的随机函数
    public static int rand1ToN(int n, int m) {
        int[] nMSys = getMSysNum(n - 1, m);
        int[] randNum = getRanMSysNumLessN(nMSys, m);
        return getNumFromMSysNum(randNum, m) + 1;
    }

    // 把value转成m进制的数
    public static int[] getMSysNum(int value, int m) {
        int[] res = new int[32];
        int index = res.length - 1;
        while (value != 0) {
            res[index--] = value % m;
```

```
            value = value / m;
        }
        return res;
    }

    // 等概率随机产生一个0~nMsys范围上的数，只不过是m进制表达的
    public static int[] getRanMSysNumLessN(int[] nMSys, int m) {
        int[] res = new int[nMSys.length];
        int start = 0;
        while (nMSys[start] == 0) {
            start++;
        }
        int index = start;
        boolean lastEqual = true;
        while (index != nMSys.length) {
            res[index] = rand1ToM(m) - 1;
            if (lastEqual) {
                if (res[index] > nMSys[index]) {
                    index = start;
                    lastEqual = true;
                    continue;
                } else {
                    lastEqual = res[index] == nMSys[index];
                }
            }
            index++;
        }
        return res;
    }

    // 把m进制的数转成10进制
    public static int getNumFromMSysNum(int[] mSysNum, int m) {
        int res = 0;
        for (int i = 0; i != mSysNum.length; i++) {
            res = res * m + mSysNum[i];
        }
        return res;
    }


    public static void main(String[] args) {
        Map<Integer, Integer> map = new HashMap<>();
        System.out.println("randM 等概率产生 [1 - n]的随机数: ");
        for (int i = 0; i < 1000000; i++) {

            // 测试方法二
            int temp = rand1ToN(4, 5);

            if(map.containsKey(temp)) {
                map.put(temp, map.get(temp) + 1);
            } else {
                map.put(temp, 1);
            }
        }

        map.forEach((key, value) -> {
            System.out.println(key + " -- 出现次数:" + value);
        });
```

```
77      }
78  }
```

```
randM 等概率产生 [1 - n]的随机数:
1 -- 出现次数:249750
2 -- 出现次数:250415
3 -- 出现次数:249921
4 -- 出现次数:249914
```

```
randM 等概率产生 [1 - n]的随机数:
1 -- 出现次数:250600
2 -- 出现次数:250113
3 -- 出现次数:249736
4 -- 出现次数:249551
```