# 题目

给你一个整数数组nums，设计算法来打乱一个没有重复元素的数组。

实现Solution class:

Solution(int[] nums)使用整数数组nums初始化对象

int[] reset()重设数组到它的初始状态并返回

int[] shuffle()返回数组随机打乱后的结果

示例：

输入

["Solution", "shuffle", "reset", "shuffle"]

[[[1, 2, 3]], [], [], []]

输出

[null, [3, 1, 2], [1, 2, 3], [1, 3, 2]]

解释

Solution solution = new Solution([1, 2, 3]);

solution.shuffle(); // 打乱数组[1,2,3]并返回结果。任何[1,2,3]的排列返回的概率应该相同。例如，返回[3, 1, 2]

solution.reset(); // 重设数组到它的初始状态 [1, 2, 3]。返回[1, 2, 3]

solution.shuffle(); // 随机返回数组[1, 2, 3]打乱后的结果。例如，返回[1, 3, 2]

提示：

1 <= nums.length <= 200

-106 <= nums[i] <= 106

nums中的所有元素都是唯一的

最多可以调用5 \* 104次reset和shuffle

# 分析

洗牌算法（Knuth shuffle算法）：对于有n个元素的数组来说，为了保证洗牌的公平性，应该要能够等概率的洗出n!种结果。

举例解释如下：

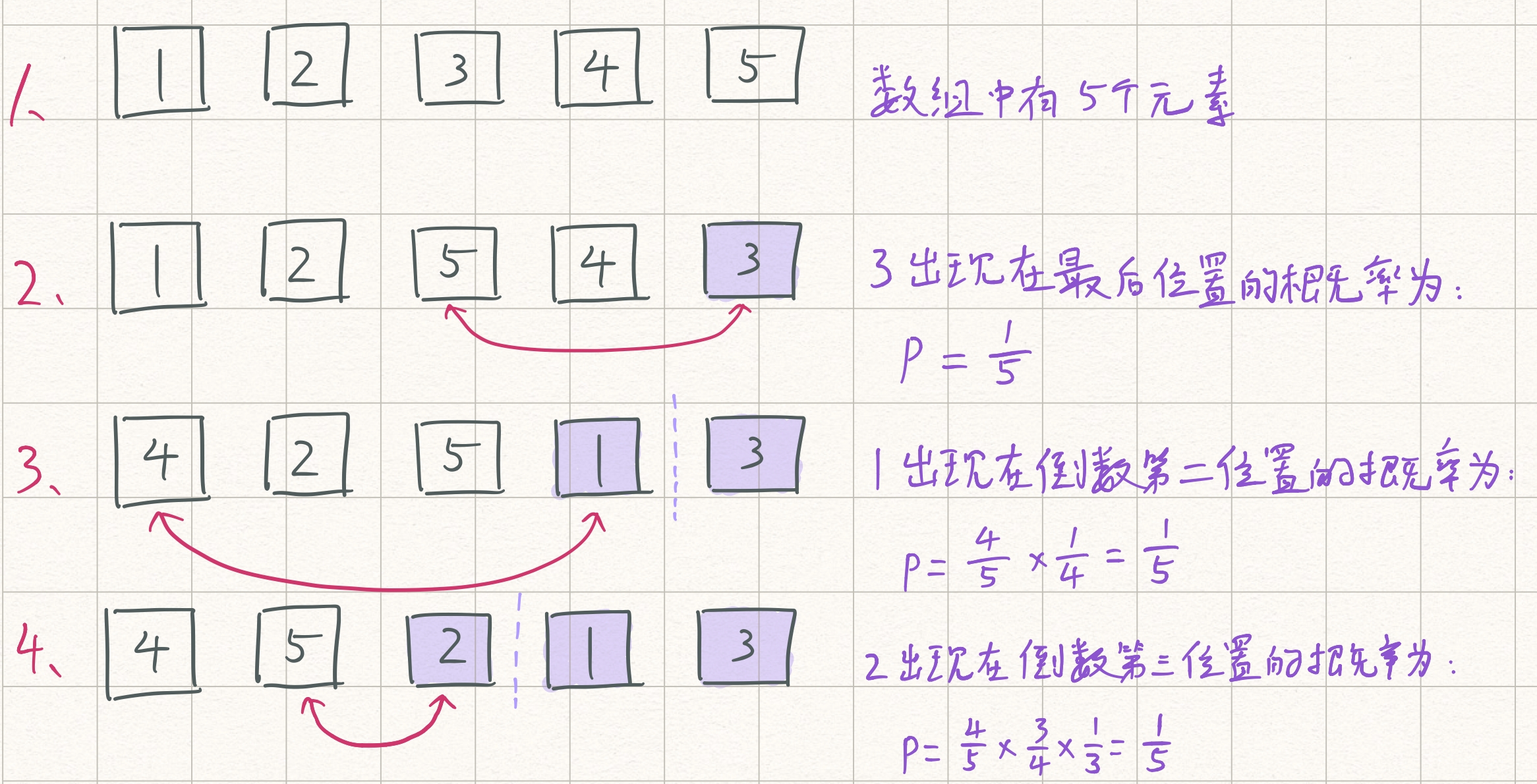
1、开始数组中有五个元素；

2、在前五个数中随机选一个数与第五个数进行交换，每个数都有五分之一的概率被交换到最后一个位置；

3、在前四个数中随机选一个数与第四个数进行交换，每个数都有五分之一的概率被交换到第四个位置；

4、在前三个数中随机选一个数与第三个数进行交换，每个数都有五分之一的概率被交换到第三个位置；

综上所述，每个数都有相等的概率被放到任意一个位置中，即每个位置中出现任意一个数的概率都是相同的。这，就是洗牌算法。



**代码：**

class Solution {

private:

vector<int> original;

public:

Solution(vector<int>& nums) {

original = nums;

}

vector<int> reset() {

return original;

}

vector<int> shuffle() {

vector<int> nums(original); //用原数组来初始化新数组

for (int i = nums.size() - 1; ~i; i -- ) //从后往前遍历

{

swap(nums[i], nums[rand() % (i + 1)]);

//rand()能随机生成0到最大随机数的任意整数

}

//rand() % (i + 1)能随机生成0到i中的任意整数

return nums;

}

};

**复杂度：**

时间复杂度：O(N)

另一种写法：

pre-C++11，rand()随机数生成，需要用Knuth shuffle算法，不能简单的rand() % nums.size()跟当前i交换。那样会导致前面的数出现的概率大于后面的数字。要用i + (rand() % (nums.size() - i))生成的index t去跟当前的i交换。

class Solution {

private:

vector<int> nums;

public:

Solution(vector<int>& nums) {

this->nums = nums;

}

/\*\* Resets the array to its original configuration and return it. \*/

vector<int> reset() {

return nums;

}

/\*\* Returns a random shuffling of the array. \*/

vector<int> shuffle() {

auto data = nums;

for (int i = 0; i < data.size(); i++) {

// 注意虽然下面注释掉的写法也能通过leetcode测试，

// 但是实际是不对的，因为其分布并不均匀

// 感兴趣的可以查看链接 http://bit.ly/3rQuFRy

// int t = rand() % data.size();

// 真正的shuffle Knuth Shuffle

int t = i + (rand() % (nums.size() - i));

swap(data[i], data[t]);

}

return data;

}

};