

随机算法之水塘抽样算法

Stars 79k

知乎 @labuladong

公众号 @labuladong

B站 @labuladong



微信搜一搜

labuladong

相关推荐：

- [一文看懂 session 和 cookie](#)
- [算法就像搭乐高：带你手撸 LFU 算法](#)

读完本文，你不仅学会了算法套路，还可以顺便去 LeetCode 上拿下如下题目：

[382. 链表随机节点](#)

[398. 随机数索引](#)

我最近在 LeetCode 上做到两道非常有意思的题目，382 和 398 题，关于水塘抽样算法（Reservoir Sampling），本质上是一种随机概率算法，解法应该说会者不难，难者不会。

我第一次见到这个算法问题是谷歌的一道算法题：给你一个未知长度的链表，请你设计一个算法，只能遍历一次，随机地返回链表中的一个节点。

这里说的随机是均匀随机（uniform random），也就是说，如果有 n 个元素，每个元素被选中的概率都是 $1/n$ ，不可以有统计意义上的偏差。

一般的想法就是，我先遍历一遍链表，得到链表的总长度 n ，再生成一个 $[1, n]$ 之间的随机数为索引，然后找到索引对应的节点，不就是一个随机的节点了吗？

但题目说了，只能遍历一次，意味着这种思路不可行。题目还可以再泛化，给一个未知长度的序列，如何在其中随机地选择 k 个元素？想要解决这个问题，就需要著名的水塘抽样算法了。

算法实现

先解决只抽取一个元素的问题，这个问题的难点在于，随机选择是「动态」的，比如说你现在你有 5 个元素，你已经随机选取了其中的某个元素 a 作为结果，但是现在再给你一个新元素 b ，你应该留着 a 还是将 b 作为结果呢，以什么逻辑选择 a 和 b 呢，怎么证明你的选择方法在概率上是公平的呢？

先说结论，当你遇到第 i 个元素时，应该有 $1/i$ 的概率选择该元素， $1 - 1/i$ 的概率保持原有的选择。看代码容易理解这个思路：

```
/* 返回链表中一个随机节点的值 */
int getRandom(ListNode head) {
    Random r = new Random();
    int i = 0, res = 0;
```

```

ListNode p = head;
// while 循环遍历链表
while (p != null) {
    // 生成一个 [0, i) 之间的整数
    // 这个整数等于 0 的概率就是 1/i
    if (r.nextInt(++i) == 0) {
        res = p.val;
    }
    p = p.next;
}
return res;
}

```

对于概率算法，代码往往都是很浅显的，但是这种问题的关键在于证明，你的算法为什么是对的？为什么每次以 $1/i$ 的概率更新结果就可以保证结果是平均随机（uniform random）？

证明：假设总共有 n 个元素，我们要的随机性无非就是每个元素被选择的概率都是 $1/n$ 对吧，那么对于第 i 个元素，它被选择的概率就是：

第 i 个元素被选择的概率是 $1/i$ ，第 $i+1$ 次不被替换的概率是 $1 - 1/(i+1)$ ，以此类推，相乘就是第 i 个元素最终被选中的概率，就是 $1/n$ 。

因此，该算法的逻辑是正确的。

同理，如果要随机选择 k 个数，只要在第 i 个元素处以 k/i 的概率选择该元素，以 $1 - k/i$ 的概率保持原有选择即可。代码如下：

```

/* 返回链表中 k 个随机节点的值 */
int[] getRandom(ListNode head, int k) {
    Random r = new Random();
    int[] res = new int[k];
    ListNode p = head;

    // 前 k 个元素先默认选上
    for (int j = 0; j < k && p != null; j++) {
        res[j] = p.val;
        p = p.next;
    }

    int i = k;
    // while 循环遍历链表
    while (p != null) {
        // 生成一个 [0, i) 之间的整数
        int j = r.nextInt(++i);
        // 这个整数小于 k 的概率就是 k/i
        if (j < k) {
            res[j] = p.val;
        }
    }
}

```

```
        p = p.next;
    }
    return res;
}
```

对于数学证明，和上面区别不大：

因为虽然每次更新选择的概率增大了 k 倍，但是选到具体第 i 个元素的概率还是要乘 $1/k$ ，也就回到了上一个推导。

拓展延伸

以上的抽样算法时间复杂度是 $O(n)$ ，但不是最优的方法，更优化的算法基于几何分布（geometric distribution），时间复杂度为 $O(k + k \log(n/k))$ 。由于涉及的数学知识比较多，这里就不列出了，有兴趣的读者可以自行搜索一下。

还有一种思路是基于「Fisher-Yates 洗牌算法」的。随机抽取 k 个元素，等价于对所有元素洗牌，然后选取前 k 个。只不过，洗牌算法需要对元素的随机访问，所以只能对数组这类支持随机存储的数据结构有效。

另外有一种思路也比较有启发意义：给每一个元素关联一个随机数，然后把每个元素插入一个容量为 k 的二叉堆（优先级队列）按照配对的随机数进行排序，最后剩下的 k 个元素也是随机的。

这个方案看起来似乎有点多此一举，因为插入二叉堆需要 $O(\log k)$ 的时间复杂度，所以整个抽样算法就需要 $O(n \log k)$ 的复杂度，还不如我们最开始的算法。但是，这种思路可以指导我们解决**加权随机抽样算法**，权重越高，被随机选中的概率相应增大，这种情况在现实生活中是很常见的，比如你不往游戏里充钱，就永远抽不到皮肤。

最后，我想说随机算法虽然不多，但其实很有技巧的，读者不妨思考两个常见且看起来很简单的问题：

1、如何对带有权重的样本进行加权随机抽取？比如给你一个数组 w ，每个元素 $w[i]$ 代表权重，请你写一个算法，按照权重随机抽取索引。比如 $w = [1, 99]$ ，算法抽到索引 0 的概率是 1%，抽到索引 1 的概率是 99%。


2、实现一个生成器类，构造函数传入一个很长的数组，请你实现 `randomGet` 方法，每次调用随机返回数组中的一个元素，多次调用不能重复返回相同索引的元素。要求不能对该数组进行任何形式的修改，且操作的时间复杂度是 $O(1)$ 。

这两个问题都是比较困难的，以后有时间我会写一写相关的文章。

刷算法，学套路，认准 **labuladong**，公众号和 [在线电子书](#) 持续更新最新文章。

本小抄即将出版，微信扫码关注公众号，后台回复「小抄」限时免费获取，回复「进群」可进刷题群一起刷题，带你搞定 **LeetCode**。



=其他语言代码=