第 9 章　排列组合与概率

排列组合常用于字符串或序列的排列和组合中，而求解排列组合的方法也比较固定：第一种是类似于动态规划的方法，即保存中间结果，依次附上新元素，产生新的中间结果；第二种是递归法，通常是在递归函数里使用 for 循环，遍历所有排列或组合的可能，然后在 for 循环语句内调用递归函数。本章所涉及的排列组合相关的问题很多都采用了上述方法。

概率论是计算机科学非常重要的基础学科之一，因为概率型面试笔试题可以综合考查求职者的思维能力、应变能力和数学能力，所以概率题也是在程序员求职过程中经常会遇到的题型。

9.1　如何拿到最多金币

难度系数：★★★★☆　　**被考查系数：**★★★★☆

题目描述：

10 个房间里放着随机数量的金币。每个房间只能进入一次，并只能在一个房间中拿金币。一个人采取如下策略：前 4 个房间只看不拿。随后的房间只要看到比前 4 个房间都多的金币数就拿。否则就拿最后一个房间的金币。编程计算这种策略拿到最多金币的概率。

分析与解答：

这道题是一个求概率的问题。由于 10 个房间里放的金币数量是随机的，因此在编程实现时首先需要生成 10 个随机数来模拟 10 个房间里的金币数量。然后判断通过这种策略是否能拿到最多的金币。如果仅通过一次模拟来求拿到最多金币的概率显然是不准确的，那么就需要进行多次模拟，通过记录模拟的次数 m，拿到最多金币的次数 n，从而可以计算出拿到最多金币的概率 n/m。显然这个概率与金币的数量以及模拟的次数有关系。模拟的次数越多越能接近真实值。下面以金币数为 1～10 的随机数、模拟次数为 1000 次为例给出实现代码：

```
/*
 ** 函数功能：      判断用指定的策略是否能拿到最多金币
 ** 函数参数：      把数组 a 看成房间，总共 n 个房间
 ** 返回值：        如果能拿到返回 1，否则返回 0
 */
function getMaxNum(a, n) {
  //随机生成 10 个房间里的金币个数
  var rand;
  for (var i = 0; i < n; i++) {
    rand = Math.floor(Math.random() * 10);
    a[i] = rand % 10 + 1;          //生成 1~10 的随机数
  }
  //找出前四个房间中最多的金币个数
  var max4 = 0;
  for (i = 0; i < 4; i++) {
```

```
        if (a[i] > max4)
            max4 = a[i];
    }
    for (i = 4; i < n - 1; i++) {
        if (a[i] > max4)
            return 1;                    //能拿到最多的金币
    }
    return 0;                            //不能拿到最多的金币
}

var a = [],
    monitorCount = 1000,
    success = 0;
for (var i = 0; i < monitorCount; i++) {
    if (getMaxNum(a, 10))
        success++;
}
console.log(success / monitorCount);
```

程序的运行结果为

```
0.421
```

运行结果分析：

运行结果与金币个数以及模拟次数都有关系，而且由于是个随机问题，因此同样的程序每次的运行结果也会不同。

9.2 如何求正整数 n 所有可能的整数组合

难度系数：★★★★☆　　**被考查系数：**★★★☆☆

题目描述：

给定一个正整数 n，求解出所有和为 n 的整数组合，要求组合按照递增方式展示，而且唯一。例如，4=1+1+1+1、1+1+2、1+3、2+2、4（即 4+0）。

分析与解答：

以数值 4 为例，和为 4 的所有的整数组合一定都小于 4（1, 2, 3, 4）。首先选择数字 1，然后用递归的方法求和为 3（即 4-1）的组合，一直递归下去直到用递归求和为 0 的组合时，所选的数字序列就是一个和为 4 的数字组合。然后第二次选择 2，接着用递归求和为 2（4-2）的组合；同理下一次选 3，然后用递归求和为 1（即 4-3）的所有组合。以此类推，直到找出所有的组合为止，实现代码如下：

```
/*
 ** 函数功能：求和为 n 的所有整数组合
 ** 输入参数：sum 为正整数，result 为组合结果，count 记录组合中的数字个数
 */
function getAllCombination(sum, result, count) {
    if (sum < 0)
```

```
        return;
    var txt = "";
    //数字的组合满足和为 sum 的条件，打印出所有组合
    if (sum == 0) {
        for (var i = 0; i < count; i++)
            txt += result[i] + " ";
        console.log("满足条件的组合：", txt);
        return;
    }
    txt = "";
    for (i = 0; i < count; i++)
        txt += result[i] + " ";
    console.log("----当前组合：", txt, "----");          //打印 debug 信息，为了便于理解
    i = (count == 0 ? 1 : result[count - 1]);           //确定组合中下一个取值
    console.log("---i=", i, "count=", count, "---");    //打印 debug 信息，为了便于理解
    for (; i <= sum;) {
        result[count++] = i;
        getAllCombination(sum - i, result, count);      //求和为 sum-i 的组合
        count--;            //递归完成后，去掉最后一个组合的数字
        i++;                //找下一个数字作为组合中的数字
    }
}

var n = 4,
    result = [];            //存储和为 n 的组合方式
//找出和为 4 的所有整数的组合
getAllCombination(n, result, 0);
```

程序的运行结果为

```
----当前组合： ----
---i=1 count=0---
----当前组合： 1 ----
---i=1 count=1---
----当前组合： 1 1 ----
---i=1 count=2---
----当前组合： 1 1 1 ----
---i=1 count=3---
满足条件的组合： 1 1 1 1
满足条件的组合： 1 1 2
----当前组合： 1 2 ----
---i=2 count=2---
满足条件的组合： 1 3
----当前组合： 2 ----
---i=2 count=1---
满足条件的组合： 2 2
----当前组合： 3 ----
---i=3 count=1---
满足条件的组合： 4
```

运行结果分析：

从上面运行结果可以看出，满足条件的组合为：{1,1,1,1}，{1,1,2 }，{1,3}，{2 ,2}，{4}。其他的为调试信息。从打印出的信息可以看出：在求和为 4 的组合中，第一步选择了 1，然后求 3（4-1）的组合也选了 1，求 2（3-1）的组合的第一步也选择了 1，以此类推，找出第一个组合为{1,1,1,1}；再通过 count--和 i++找出最后两个数字 1 与 1 的另外一种组合 2，最后三个数字的另外一种组合 3；接下来用同样的方法分别选择 2、3 作为组合的第一个数字，就可以得到以上结果。

代码 i=(count==0 ? 1 : result[count-1])用来保证组合中的下一个数字一定不会小于前一个数字，从而保证了组合的递增性。如果不要求递增（如把{1, 1, 2}和{2, 1, 1}看作两种组合），那么把上面一行代码改成 i=1 即可。

9.3 如何用一个随机函数得到另外一个随机函数

难度系数：★★★★☆　　被考查系数：★★★☆☆

题目描述：

有一个函数 fun1()能返回 0 和 1 两个值，并且返回 0 和 1 的概率都是 1/2，怎么利用这个函数得到另一个函数 fun2()，使 fun2()也只能返回 0 和 1，且返回 0 的概率为 1/4，返回 1 的概率为 3/4。

分析与解答：

函数 fun1()得到 1 与 0 的概率都为 1/2。因此，可以调用两次 fun1()，分别生成两个值 a1 与 a2，用这两个数组成一个二进制 a2a1，它取值的可能性为 00，01，10 和 11，并且得到每个值的概率都为(1/2)×(1/2)=1/4。因此，如果得到的结果为 00，则返回 0（概率为 1/4），其他情况则返回 1（概率为 3/4）。实现代码如下：

```
//返回 0 和 1 的概率都为 1/2
function fun1() {
  return Math.floor(Math.random() * 2) % 2;
}
//返回 0 的概率为 1/4，返回 1 的概率为 3/4
function fun2() {
  var a1 = fun1(),
    a2 = fun1(),
    tmp = a1;
  tmp |= (a2 << 1);
  if (tmp == 0)
    return 0;
  return 1;
}

var arr = [];
for (var i = 0; i < 16; i++)
  arr.push(fun2());
console.log(arr.join(" "));
arr = [];
```

```
for (i = 0; i < 16; i++)
   arr.push(func2());
console.log(arr.join(" "));
```

程序的运行结果为

```
1 1 1 0 1 1 0 1 1 0 1 0 1 1 1 0 1
1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 0
```

由于结果是随机的，调用的次数越大，返回的结果越接近 1/4 与 3/4。

9.4 如何等概率地从大小为 n 的数组中选取 m 个整数

难度系数：★★★★☆ **被考查系数：★★★☆☆**

题目描述：

随机地从大小为 n 的数组中选取 m 个整数，要求每个元素被选中的概率相等。

分析与解答：

从 n个数中随机选出一个数的概率为 1/n，然后在剩下的 n-1 个数中再随机找出一个数的概率也为 1/n（第一次没选中这个数的概率为 (n-1)/n，第二次选中这个数的概率为 1/(n-1)，因此，随机选出第二个数的概率为(n-1)/n)×(1/(n-1))=1/n，依此类推，在剩下的 k 个数中随机选出一个元素的概率都为 1/n。这个算法的思路为：首先从包含 n 个元素的数组中随机选出一个元素，然后把这个选中的数字与数组第一个元素交换，接着从数组后面的 n-1 个数字中随机选出一个数字与数组第二个元素交换，依此类推，直到选出 m 个数字为止，数组前 m 个数字就是随机选出来的 m 个数字，且它们被选中的概率相等。

以数组[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]为例，实现代码如下：

```
function getRandomM(a, n, m) {
   if (n <= 0 || n < m) {
      return;
   }
   var j, rand, tmp;
   for (var i = 0; i < m; ++i) {
      rand = Math.floor(Math.random() * (n - i));
      j = i + rand;                    //获取 i 到 n-1 之间的随机数
      //随机选出的元素放到数组的前面
      tmp = a[i];
      a[i] = a[j];
      a[j] = tmp;
   }
}

var a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
   n = a.length,
   m = 6,
   txt = "";
getRandomM(a, n, m);
```

```
for (i = 0; i < m; ++i)
    txt += a[i] + " ";
console.log(txt);
```

程序的运行结果为

```
1  8  9  7  2  4
```

算法性能分析：

这个算法的时间复杂度为 O(m)。

9.5 如何计算 1、2、5 这三个数使其和为 100 的组合个数

难度系数：★★★★☆ **被考查系数：★★★★☆**

题目描述：

求出用 1、2、5 这三个数的不同个数组合的和为 100 的组合个数。为了更好地理解题目的意思，下面给出几组可能的组合：100 个 1、0 个 2、0 个 5 的和为 100；50 个 1、25 个 2、0 个 5 的和也是 100；50 个 1、20 个 2、2 个 5 的和也为 100。

分析与解答：

方法一：蛮力法

最简单的方法就是对所有的组合进行尝试，然后判断组合的结果是否满足和为 100，这些组合有如下限制：1 的个数最多为 100 个，2 的个数最多为 50 个，5 的个数最多为 20 个。实现思路为：遍历所有可能的组合包含 1 的个数 x（0≤x≤100），2 的个数 y（0≤y≤50），5 的个数 z（0≤z≤20），再判断 x+2×y+5×z 是否等于 100，如果等于 100，则满足条件。实现的代码如下：

```
function combinationCount1(n) {
    var count = 0,
        num1 = n,                    //1 最多的个数
        num2 = n / 2,                //2 最多的个数
        num5 = n / 5;                //5 最多的个数
    for (var x = 0; x <= num1; x++)
        for (var y = 0; y <= num2; y++)
            for (var z = 0; z <= num5; z++) {
                if (x + 2 * y + 5 * z == n)          //满足条件
                    count++;
            }
    return count;
}
console.log(combinationCount1(100));
```

程序的运行结果为

```
541
```

算法性能分析：

这个算法循环的次数为 101×51×21。

方法二：数字规律法

针对这种数学公式的运算，一般都可以通过找出运算规律来简化运算过程。对于本题而言，对 x+2y+5z=100 进行变换可以得到 x+5z=100-2y。从这个表达式可以看出，x+5z 是偶数且 x+5z≤100。因此，求满足 x+2y+5z=100 的组合个数就可以转换为求满足“x+5z 是偶数且 x+5z≤100”的个数。可以通过对 z 的所有可能取值（0≤z≤20）进行遍历从而计算满足条件的 x 的值。

当 z=0 时，x 的取值为 0, 2, 4, …, 100（100 以内的所有偶数），个数为(100+2)/2。

当 z=1 时，x 的取值为 1, 3, 5, …, 95（95 以内的所有奇数），个数为(95+2)/2。

当 z=2 时，x 的取值为 0, 2, 4, …, 90（90 以内的所有偶数），个数为(90+2)/2。

当 z=3 时，x 的取值为 1, 3, 5, …, 85（85 以内的所有奇数），个数为(85+2)/2。

……

当 z=19 时，x 的取值为 5, 3, 1（5 以内的所有奇数），个数为(5+2)/2。

当 z=20 时，x 的取值为 0（0 以内的所有偶数），个数为(0+2)/2。

实现的代码如下：

```
function combinationCount2(n) {
  var count = 0;
  for (var m = 0; m <= n; m += 5) {
    count += Math.floor((m + 2) / 2);
  }
  return count;
}
```

算法性能分析：

这个算法循环的次数为 21。

9.6 如何判断有几盏灯泡还亮着

难度系数：★★★★☆ **被考查系数：★★★★★**

题目描述：

100 个灯泡排成一排，第一轮将所有灯泡打开；第二轮每隔一个灯泡关掉一个，即排在偶数位置的灯泡被关掉，第三轮每隔两盏灯泡，将开着的灯泡关掉，关掉的灯泡打开。依此类推，第 100 轮结束的时候，还有几盏灯泡亮着？

分析与解答：

1）对于每盏灯，当拉动的次数是奇数时，灯就是亮着的；当拉动的次数是偶数时，灯就是关着的。

2）每盏灯拉动的次数与它的编号所含约数的个数有关，它的编号有几个约数，这盏灯就被拉动几次。

3）1～100 这 100 个数中有哪几个数的约数个数是奇数？

一个数的约数都是成对出现的，只有完全平方数的约数个数才是奇数。所以，这 100 盏灯中有 10 盏灯是亮着的，它们的编号分别是：1、4、9、16、25、36、49、64、81、100。实现代码如下：

```
function factorIsOdd(a) {
    var total = 0;
    for (var i = 1; i <= a; i++) {
        if (a % i == 0)
            total++;
    }
    if (total % 2 == 1)
        return 1;
    return 0;
}
function totalCount(num, n) {
    var count = 0;
    for (var i = 0; i < n; i++) {
        //判断因子数是不是奇数，奇数（灯亮）则加 1
        if (factorIsOdd(num[i])) {
            console.log("亮着的灯的编号是：", num[i]);
            count++;
        }
    }
    return count;
}

var num = new Array(101)
    .join("0").split("")
    .map(function (value, index) {
        return index + 1;
    });
var count = totalCount(num, 100);
console.log("最后总共有", count, "盏灯亮着。");
```

程序的运行结果为

```
亮着的灯的编号是：1
亮着的灯的编号是：4
亮着的灯的编号是：9
亮着的灯的编号是：16
亮着的灯的编号是：25
亮着的灯的编号是：36
亮着的灯的编号是：49
亮着的灯的编号是：64
亮着的灯的编号是：81
亮着的灯的编号是：100
最后总共有 10 盏灯亮着。
```