第七章 四个实验

1. 循环赛日程表

```
设有n=2^k个运动员要进行网球循环赛。现要设计一个满足以下要求的比赛日程表:
(1)每个选手必须与其他n-1个选手各赛一次;
(2)每个选手一天只能赛一次;
(3)循环赛一共进行n-1天。
分析:按照上面的要求,可以将比赛表设计成一个n行n-1列的二维表,其中第 i 行第 j 列的元素表示和第 i 个选手在第 j 天比赛的选手号。
日程表如下:(行:选手号、列:第几天)
```

1	2	3	4	5	6	7	8
2	1	4	3	6	5	8	7
3	4	1	2	7	8	5	6
4	3	2	1	8	7	6	5
5	6	7	8	1	2	3	4
6	5	8	7	2	1	4	3
7	8	5	6	3	4	1	2
8	7	6	5	4	3	2	1

规律:

```
左上角元素手动赋值
a[0][0] = 1;
a[0][1] = 2;
a[1][0] = 2;
a[1][1] = 1;
```

```
左下角元素 = 左上角元素 + 2^(k-1)(当前构建的矩阵)
```

```
右上角 = 左下角元素
```

```
右下角 = 左上角的元素
```

循环构建:

```
const fun = (k) => {
  // 创建一个2^k阶矩阵
  let table = new Array(Math.pow(2,k));
  const len = table.length;
  for(let i = 0; i < len; i++){
    table[i] = new Array(Math.pow(2,k));
  }
  // 赋值初始化左上角数字
  a[0][0] = 1;
  a[0][1] = 2;
  a[1][0] = 2;
  a[1][1] = 1;
  // 按照规则打表
  const write = (k)=>{
    let row = Math.pow(2,1);
    let col = Math.pow(2,1);
    while(row <= Math.pow(2,k) && col <= Math.pow(2,k)){
      // 打表左下角
      for(let i = row; i < row * 2; i++){
        for(let j = 0; j < col; j++){
          table[i][j] = table[i - row][j] + row;
        }
      }
      // 打表右上角
      for(let i = 0; i < row; i++){
        for(let j = col; j < col * 2; col++){
          table[i][j] = table[i + row][j - col];
        }
      }
      // 打表右下角
      for(let i = row; i < row * 2; i++){
        for(let j = col; j < col * 2; j++){
          table[i][j] = table[i -row][j - col];
        }
      }
      row *= 2;
      col *= 2;
    }
  }
  // 打印表
  table.map(rows=>{
    rows.map(item=>{
      console.log(item + " ")
    })
    console.log("\n")
  })
}
```

2. 求三个数的最小公倍数

   1️⃣蛮力法

   从三数最大值开始一路次方，当第一个能被三数整除的就是三数最小公倍数

```
const min_common_multiple = (num1, num2, num3) => {
  const max = Math.max(num1,num2,num3);
  let i = 1;
  while(true){
    const target = Math.pow(max,i);
    if(target % num1 === 0 && target % num2 === 0 && target % num3 === 0){
      return target;
    }
  }
}
```

   2️⃣利用最大公约数求最小公倍数

```
const gcd = (num1, num2) => num1 % num2 === 0 ? num2 : gcd(num2, num1 % num2)
const common_multiple = (num1, num2, num3) => num1 * num2 / gcd(num1, num2) * num3 / gcd(num1 * num2 / gcd(num1, num2),num3)
```

3. 猴子选大王

   17个猴子围成一圈，从某个开始报数1-2-3-1-2-3-......报“3”的猴子就被淘汰，游戏一直进行到圈内只剩一只猴子它就是猴大王了。

```
const monkeyKing = (num, no) => {
  let count = 0;
  const array = new Array(num).fill(1);
  let i = 0;
  while(true){
    i ++
    if(count === num - 1){
      return i;
    }
    if(i % no === 0){
      array[i] = 0;
      count ++
    }
    i = i % num
  }
}
```

4. 最大子段和问题

   输入一个整型数组，数组中的一个或连续多个整数组成一个子数组。求所有子数组的和的最大值。

要求时间复杂度为O(n)。

```
输入: nums = [-2,1,-3,4,-1,2,1,-5,4]
输出: 6
解释: 连续子数组 [4,-1,2,1] 的和最大, 为 6。
```

方法一：动态规划

思路和算法

假设 $nums$ 数组的长度是 n，下标从 0 到 $n-1$。

我们用 $f(i)$ 代表以第 i 个数结尾的「连续子数组的最大和」，那么很显然我们要求的答案就是：

$$\max_{0 \le i \le n-1} \{f(i)\}$$

因此我们只需要求出每个位置的 $f(i)$，然后返回 f 数组中的最大值即可。那么我们如何求 $f(i)$ 呢？我们可以考虑 $nums[i]$ 单独成为一段还是加入 $f(i-1)$ 对应的那一段，这取决于 $nums[i]$ 和 $f(i-1)+nums[i]$ 的大小，我们希望获得一个比较大的，于是可以写出这样的动态规划转移方程：

$$f(i) = \max\{f(i-1) + nums[i], nums[i]\}$$

不难给出一个时间复杂度 $O(n)$、空间复杂度 $O(n)$ 的实现，即用一个 f 数组来保存 $f(i)$ 的值，用一个循环求出所有 $f(i)$。考虑到 $f(i)$ 只和 $f(i-1)$ 相关，于是我们可以只用一个变量 pre 来维护对于当前 $f(i)$ 的 $f(i-1)$ 的值是多少，从而让空间复杂度降低到 $O(1)$，这有点类似「滚动数组」的思想。

```javascript
var maxSubArray = function(nums) {
    let pre = 0, maxAns = nums[0];
    nums.map(x => {
        pre = Math.max(pre + x, x);
        maxAns = Math.max(maxAns, pre);
    });
    return maxAns;
};
```