题目一

给定一个数组arr，求差值为k的去重数字对。

例如： arr=[2,6,8,10] k=2 返回`[[6,8], [8,10]]`

思路：

1. 将数组加入哈希set中，遍历set判断cur+k在不在set中

```
function subvalueEqualK(arr, k) {
    const set = new Set();
    const res = [];
    for (let i = 0; i < arr.length; i++) {
        set.add(arr[i]);
    }
    for (let cur of set) {
        if (set.has(cur+k)) {
            res.push([cur, cur+k]);
        }
    }
    return res;
}
```

题目二

给一个包含n个整数元素的集合a，一个包含m个整数元素的集合b。

定义magic操作为，从一个集合中取出一个元素，放到另一个集合里，且操作过后每个集合的平均值都大大于于操作前。

注意以下两点：

1. 不可以把一个集合的元素取空，这样就没有平均值了
2. 值为x的元素从集合b取出放入集合a，但集合a中已经有值为x的元素，则a的平均值不变（因为集合元素不会重复），b的平均值可能会改变（因为x被取出了）

问最多可以进行多少次magic操作?

思路：

1. 分别计算集合a和b的平均值，avgA和avgB;
2. magic操作有以下几种可能：
    1. 如果avgA = avgB，无论如何操作都不符合；
    2. 从较小平均值(假设为avgA)中取值x放入b中
        1. 如果x < avgA < avgB会导致avgA变大、avgB变小；
        2. 如果avgA < x < avgB会导致avgA变小、avgB变小；
        3. 如果avgA < avgB < x会导致avgA变小、avgB变大；

3. 从较大平均值(假设为avgB)中取值x放入a中
    1. 如果x < avgA < avgB会导致avgA变小、avgB变大；
    2. 如果avgA < x < avgB会导致avgA变大、avgB变大；
    3. 如果avgA < avgB < x会导致avgA变大、avgB变小；
3. 只有从较大平均值的集合中取值为两个集合平均值之间的数才符合条件；
4. 将两个集合从小到大排序，拿满足要求最小的数字，会使得较大的平均值最大幅度上升，较小的平均值最小幅度上升，使得magic的次数尽量多；
5. 可证从较大平均值中选两个平均值间的数不会导致较小的平均值大于较大平均值，因此只需要一直从较大平均值的集合取值；

```
function magicOp(arr1, arr2) {
    let sum1 = 0;
    for (let i = 0; i < arr1.length; i++) {
        sum1 += arr1[i]
    }
    let sum2 = 0;
    for (let i = 0; i < arr2.length; i++) {
        sum2 += arr2[i]
    }
    // 如果平均值相等返回0
    if (avg(sum1, arr1.length) === avg(sum2, arr2.length)) {
        return 0;
    }
    let arrMore = null;
    let arrLess = null;
    let sumMore = 0;
    let sumless = 0;

    if (avg(sum1, arr1.length) > avg(sum2, arr2.length)) {
        arrMore = arr1;
        arrLess = arr2;
        sumMore = sum1;
        sumless = sum2;
    } else {
        arrMore = arr2;
        arrLess = arr1;
        sumMore = sum2;
        sumless = sum1;
    }
    arrMore.sort();
    let set = new Set();
    for (let i = 0; i < arrLess.length; i++) {
        set.add(arrLess[i])
    }
    let moreSize = arrMore.length;
    let lessSize = arrLess.length;
    let res = 0;
    for (let i = 0; i < arrMore.length; i++) {
        let cur = arrMore[i]
        if (cur < avg(sumMore, moreSize) && cur > avg(sumless, lessSize) && !set.has(cur)) {
            sumMore -= cur;
            moreSize -= 1;
            sumless += cur;
            lessSize++;
            set.add(cur);
            res++;
        }
    }
    return res;
```

```
}

function avg(sum, length) {
    return sum / length
}
```

题目三

将给定的数转换为字符串，原则如下：

1对应 a，2对应b，…..26对应z，例如12258可以转换为"abbeh", "aveh", "abyh", "lbeh" and "lyh"，个数为5，

编写一个函数，给出可以转换的不同字符串的个数

思路：

1. 将数字转化为字符串数组；
2. 从左往右尝试（假设当前下标i，值cur）：
    1. 如果i越界返回1；
    2. 如果cur等于0返回0；
    3. 如果i等于arr.length - 1返回1
    4. 如果当前cur和下一个位置拼成的数小于27，返回process(i+1) + process(i+2)，否则返回process(i+1);

```
function NumsToStringWays(num) {
    const arr = num.toString().split("");
    function process(index) {
        let cur = parseInt(arr[index])
        if (index === arr.length) {
            return 1
        }
        if (cur === 0) {
            return 0;
        }
        if (index === arr.length - 1) {
            return 1;
        }
        let res = 0;
        if (cur * 10 + parseInt(arr[index+1]) < 27) {
            res = process(index + 1) + process(index + 2)
        } else {
            res = process(index + 1)
        }

        return res;
    }
    return process(0)
}

// 动态规划
function NumsToStringWays2(num) {
    const arr = num.toString().split("");
    const dp = new Array(arr.length+1);
    dp[arr.length] = 1
    dp[arr.length - 1] = arr[arr.length - 1] === "0" ? 0 : 1;

    for (let i = arr.length - 2; i >= 0; i--) {
```

```
        let cur = parseInt(arr[i])
        if (cur === 0) {
            dp[i] = 0
        } else {
            if (cur * 10 +  parseInt(arr[i+1]) < 27) {
                dp[i] = dp[i + 1] + dp[i+2]
            } else {
                dp[i] = dp[i+1]
            }
        }
    }

    return dp[0]
}
```

题目四

一个合法的括号匹配序列有以下定义:

1. 空串""是一个合法的括号匹配序列
2. 如果"X"和"Y"都是合法的括号匹配序列,"XY"也是一个合法的括号匹配序列
3. 如果"X"是一个合法的括号匹配序列,那么"(X)"也是一个合法的括号匹配序列
4. 每个合法的括号序列都可以由以上规则生成。

例如: "","()","()()","((()))"都是合法的括号序列

对于一个合法的括号序列我们又有以下定义它的深度:

1. 空串""的深度是0
2. 如果字符串"X"的深度是x,字符串"Y"的深度是y,那么字符串"XY"的深度为max(x,y) 3、如果"X"的深度是x,那么字符串"(X)"的深度是x+1

例如: "()()()"的深度是1,"((()))"的深度是3。牛牛现在给你一个合法的括号序列,需要你计算出其深度。

思路:

1. 用变量leftCount辅助，如果当前为“（”leftCount加一，如果为“）”leftCount减一；
2. 深度为leftCount最大值

```
function parenthesesDeep(str) {
    const arr = str.split("");
    let count = 0;
    let res= 0;

    for (let i = 0; i < arr.length; i++) {
        if (arr[i] === "(") {
            count++;
            res = Math.max(res, count)
        } else {
            count--;
        }
    }

    return res;
}
```

拓展

找到最长的有效括号子串的长度

例如：（）） （） （（）） （））） （ 为8

思路：

1. 遍历字符串数组，用dp记录以当前下标为结尾的最长有效括号子串的长度；
2. 如果当前为index，值为cur。如果cur===“（”，此时`dp[index] = 0`;
3. 如果cur === “）”，判断`index- 1 - dp[index-1]`的值：
   - 如果值为“）”则`dp[index] = dp[index-1]`;
   - 如果值为“（”则`dp[index] = dp[index-1] + 2 + dp[index-2-dp[index-1]]`;

```
function parenthesesMaxValidLength(str) {
    const arr = str.split("");
    const dp = new Array(arr.length).fill(0);
    let res = 0
    for (let i = 1; i < arr.length; i++) {
        if (arr[i] === ")") {
            let prevIndex = i - 1 - dp[i-1];
            if (prevIndex >= 0 &&  arr[prevIndex] === "(") {
                dp[i] = dp[i-1] + 2 + (prevIndex - 1 < 0 ? 0 : dp[prevIndex - 1])
            }
        }
        res = Math.max(res, dp[i])
    }
    return res
}
```

题目五

请编写一个程序，对一个栈里的整型数据，按升序进行排序（即排序前，栈里的数据是无序的，排序后最大元素位于栈顶），要求最多只能使用一个额外的栈存放临时数据，但不得将元素复制到别的数据结构中。

思路：

1. 准备辅助栈helpStack，依次将stack中栈顶元素cur弹出，如果helpStack为空cur直接入栈，否则比较cur和helpStack栈顶元素helpCur的大小：
   - cur <= helpCur，cur直接压栈；
   - cur > helpCur，helpStack弹出元素压入stack中，直到helpStack栈顶大于cur或者helpStack为空，将cur放入；
2. 当stack为空，依次弹出helpStack压入stack；

```
function stackSortStack(stack) {
    const helpStack = [];
    while (stack.length > 0) {
        let cur = stack.pop();
        while (helpStack.length > 0 && helpStack[helpStack.length-1] < cur) {
            stack.push(helpStack.pop());
        }
        helpStack.push(cur);
    }
    while (helpStack.length > 0) {
        stack.push(helpStack.pop())
    }
```

```
    return stack;
}
```

题目六

二叉树每个结点都有一个int型权值，给定一棵二叉树，要求计算出从根结点到叶结点的所有路径中，权值和最大的值为多少。

思路：

1. 权值的最大值为当前node的权值加上Max(以左子节点为头的最大权值和，以右子节点为头的最大权值和)；
2. 如果head为null返回0（base case）；

```
function MaxSumInTree(head) {
    function process(head) {
        if (head === null) {
            return 0;
        }
        let leftMaxSum = process(head.left);
        let rightMaxSum = process(head.right);
        return head.val + Math.max(leftMaxSum, rightMaxSum);
    }

    return process(head);
}
```