1

实现一个函数`reverse(a, n)`，反转一个含有 n 个整数的数组 a（直接在数组a上操作，元素交换次数尽可能少，不能使用js Array 类内置属性和方法）。

```
var x = [0, 1, 2, 3]
reverse(x, 4) // x = [3, 2, 1, 0]

var y = [1, 2, 3, 4, 1]
reverse(y, 5) // y = [1, 4, 3, 2, 1]
```

2

实现一个函数`countLongest(tree)`，输入一棵二叉树，返回二叉树中距离最长的两个叶子节点之间的距离。

```
var tree1 = {
    value: 1,
    left: {
        value: 2
    },
    right: {
        value: 3
    }
}
countLongest(tree1) // 2

var tree2 = {
    value: 1,
    left: {
        value: 2,
        left: {
            value: 3,
            left: {
                value: 6
            }
        },
        right: {
            value: 4
        }
    },
    right: {
        value: 5
    }
}
countLongest(tree2) // 4
```

3

在前端开发中，通常会把多个js文件合并成一个文件，以减少网络请求次数，达到优化加载速度的目的，但是当文件之间存在依赖关系时，对js合并的顺序，会有一定的要求，比如 A.js 依赖了 B.js，那打包后的文件，B.js 需要排在 A.js 的前面。

实现一个函数`resolve(tree)`，根据js的依赖关系树 tree，输出合理的打包顺序的数组（结果可能不唯一，输出其中一种即可）。

样例

```
var tree1 = {
    name: 'main.js',
    require: [{
        name: 'A.js'
    }, {
        name: 'B.js'
    }]
}
resolve(tree1) // ['A.js', 'B.js', 'main.js']

var tree2 = {
    name: 'page.js',
    require: [{
        name: 'A.js',
        require: [{
            name: 'B.js',
            require: [{
                name: 'C.js'
            }]
        }]
    }, {
        name: 'D.js',
        require: [{
            name: 'C.js'
        }, {
            name: 'E.js'
        }]
    }]
}
resolve(tree2) // ['C.js', 'E.js', 'D.js', 'B.js', 'A.js', 'page.js']
```

4

给定一个整数数组 a，实现一个函数`countMax(a)`，计算出从 a 中选择出多个不相邻元素组成最大的和是多少。

样例

```
var x = [1, 4, 5, 3]
countMax(x) // 7

var y = [3, 12, 6, 2, 4]
countMax(y) // 16
```