
ELBRUS/
BOOT
CAMP

Алгоритмы и Структуры Данных

- Оценка сложности
- String
- Array
- Hash Table
- Linked List
- Permutations
- Sorting

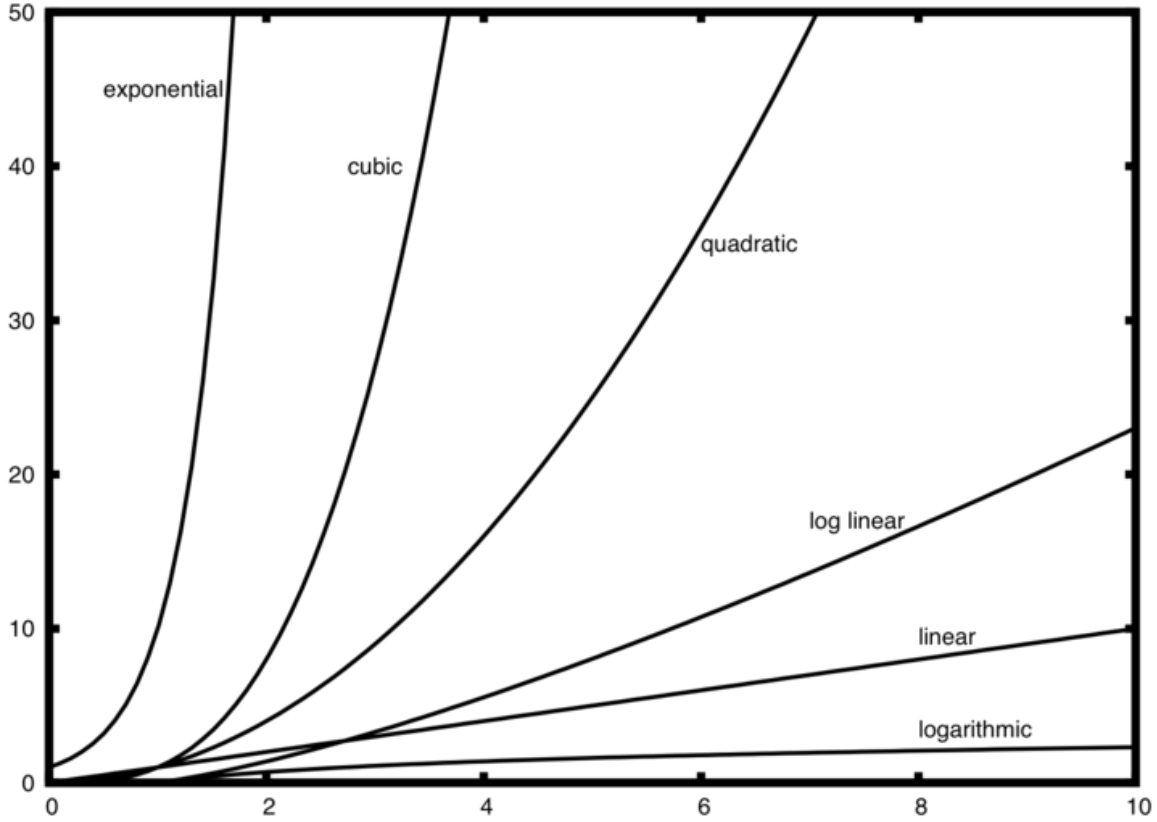
1. Оценка сложности

По времени выполнения O(N)

- O(n) — линейная сложность
- O(log n) — логарифмическая сложность
- $O(n^2)$ — квадратичная сложность
- O(1) — не зависит от входных данных

размер / сложность	10	20	30	40	50	60
n	0,00001 сек.	0,00002 сек.	0,00003 сек.	0,00004 сек.	0,00005 сек.	0,00005 сек.
n^2	0,0001 сек.	0,0004 сек.	0,0009 сек.	0,0016 сек.	0,0025 сек.	0,0036 сек.
n^3	0,001 сек.	0,008 сек.	0,027 сек.	0,064 сек.	0,125 сек.	0,216 сек.
n^5	0,1 сек.	3,2 сек.	24,3 сек.	1,7 минут	5,2 минут	13 минут
2^n	0,0001 сек.	1 сек.	17,9 минут	12,7 дней	35,7 веков	366 веков
3^n	0,059 сек.	58 минут	6,5 лет	3855 веков	$2x10^8$ веков	$1,3x10^{13}$ веков

По времени выполнения O(N)



По использованию памяти O(1)

- O(n) — продублировали все входные данные в отдельной структуре
- O(1) — не использовали дополнительную память

2. Задачи на String

1. Палиндром

```
var str = 'str’
console.log(str[0])
// s


/** * @param {string} s *
@return {boolean} */

var isPalindrome = function(s) {

};
```

1. Палиндром

```
var isPalindrome = function(s) {
	s = s.toLowerCase();
	var len = s.length;
	for (var i = 0; i < len/2; i++) {
		if (s[i] !== s[len - 1 - i]) {
			return false;
		}
	}
	return true;
};
console.log(isPalindrome("AB"));
console.log(isPalindrome("ABA"));
console.log(isPalindrome("AA"));
console.log(isPalindrome("A  a"));
console.log(isPalindrome("PAL LA P"));
```

1. Палиндром

```
var isPalindrome = function(s) {
    var len = s.length;    var left = 0;    var right = len-1;
    while(left <= right) {
        if(s[left] == ' ') {
            left++;
            continue;
        }
        if(s[right] == ' '){
            right--;
            continue;
        }
        if (s[left].toLowerCase() !== s[right].toLowerCase())
            return false;
        left++;     right--;
    }   return true;};
```

2. Перевернуть строку

Example 1:
Input: "hello"
Output: "olleh"

Example 2:
Input: "A man, a plan, a canal: Panama"
Output: "amanaP :lanac a ,nalp a ,nam A"

3. Самое часто встречающее слово в тексте

Input: paragraph = "Bob hit a ball, the hit BALL flew far after it was hit. "

Output: hit

4. Свертка строк

Input: ["a","a","b","b","c","c","c"]

Output: ["a","2","b","2","c","3"]

5. Проверка правильности расстановки скобок

Input: "()"
Output: true

Input: "()[]{}"
Output: true

Input: "(]"
Output: false

Input: "(])"
Output: false

Input: "{[]}"
Output: true

3. Задачи на Массивы

Массивы:

```
var arr = [];
arr[0] = 1
console.log(arr.length); //(0)
console.log(arr[0]); //(1)
console.log(arr[1]); //(2)

arr[2] = 'A'
console.log(arr.length); //(3)
console.log(arr[0]); //(4)
console.log(arr[1]); //(5)
console.log(arr[2]); //(6)
```

1. Склеить два сортированных списка:

```
console.log([2, 7, 6].sort())
```

```
var mergeArrays =
function(arr1, arr2) {
  mergedArray = []
  return mergedArray;
}
```

1. Склеить два сортированных списка:

Решение 1

```
var mergeArrays = function(arr1, arr2) {
    mergedArray = []

    for(i=0; i<arr1.length; i++){
        mergedArray[i] = arr1[i];
    }

    for(i=0; i<arr2.length; i++){
        mergedArray[arr1.length+i] = arr2[i];
    }
    return mergedArray.sort();
}
console.log(mergeArrays([1, 4], [2, 3]))
```

1. Склеить два сортированных списка:

Решение 2

```
var mergeArrays = function(arr1, arr2) {
    for(i=0; i<arr2.length; i++) {
        arr1[arr1.length+i] = arr2[i];
}
    return arr1.sort();
}

console.log(mergeArrays([1, 4], [2, 3]))
```

[1, 2, 3, 4, <1 empty item>]

1. Склеить два сортированных списка:

Решение 3

```
var mergeArrays = function(arr1, arr2) {
    let arrLength = arr1.length
    for(i=0; i<arr2.length; i++) {
        arr1[arr1.length+i] = arr2[i];
        arr1[arrLength] = arr2[i];
}
    return arr1.sort();
}

console.log(mergeArrays([1, 4], [2, 3]))
```

[1, 2, 3, 4]

1. Склеить два сортированных списка:

Решение 4

```
var mergeArrays = function(arr1, arr2) {
    var mergedArray = []
    let i = 0, j = 0, z = 0;

    while(i < arr1.length && j < arr2.length) {
        if(arr1[i] < arr2[j]){
            mergedArray[z] = arr1[i];
            i++;
        } else {
            mergedArray[z] = arr2[j];
            j++;
        }
        z++;
}return mergedArray;}
```

[1, 2, 3]

1. Склеить два сортированных списка:

Решение 4

```
..
    for(;i<arr1.length;i++, z++){
        mergedArray[z] = arr1[i];
    }

    for(;j<arr2.length;j++, z++){
        mergedArray[z] = arr2[j];
    }

    return mergedArray;
}
```

1. Склеить два сортированных списка:

Решение 5

```
var mergeArrays = function(arr1, arr2) {
    let i = arr1.length-1, j = arr2.length-1, z = i+j+1;

    while(i>=0 && j>=0){
        if(arr1[i] > arr2[j]){
        arr1[z] = arr1[i];
        i--;
    }else{
        arr1[z] = arr2[j];
        j--;
    }
    z--;
    }
    for(;j>=0;j--, z--){ arr1[z] = arr2[j]; }

return arr1;}
```

2. Найти в сортированном списке два элемента сумма которых равна target

Input: numbers = [2,7,11,15], target = 9
Output: [0,1]

```
/**
* @param {number[]} numbers
* @param {number} target
* @return {number[]}
*/
var twoSum = function(numbers, target) {
};
```

2. Найти в сортированном списке два элемента сумма которых равна равна заданному числу

```
var twoSumHash = function(numbers, target) {
    map = {};
    for(i=0;i<=numbers.length-1;i++) {
        map[numbers[i]] = i
    }

    for(i=0; i < numbers.length-1; i++) {
        let numberToFind = target - numbers[i]
        if(map[numberToFind] != null)
             && map[numberToFind] != i
{           return [map[numberToFind], i]
        }
}return [];};
console.log(twoSumHash([2,3], 4))
```

2. Найти в сортированном списке два элемента сумма которых равна заданному числу

```
var twoSum = function(numbers, target) {
    let i = 0; j = numbers.length-1;
    while(i <= j) {
    summ = numbers[i]+numbers[j]
    if(summ==target){
        return [i, j]
    } else if(summ < target) {
        i++
    }else{
        j--
    }
    }
};
```

3. Убрать дубликаты из сортированного списка:

Input: [1, 2, 2, 2, 3, 3]

Output: [1, 2, 3]

4. Даны числа в массиве 1..n $1 \le a[i] \le n$
Найти все пропущенные числа в массиве:

Input: [4,3,2,7,8,2,3,1]

Output: [5, 6]

5. Сказать есть ли в массиве сумма последовательных элементов = target

Input: [3,1, 3, 2] **Target:** 6

Output: true

Input: [3,1, 5, 3, 2] **Target:** 6

Output: false

4. Задачи на Hash Map
Hash Table, …
Javascript - Objects…

Hash Table:

```
table = {"key1" : "value1"};

console.log(table["key1"])
console.log(table["key2"])

>> value1
>> undefind

table[2] = "value2"
console.log(table["key1"])
console.log(table[2])

>> value1
>> value2
```

1. Посчитать число простых чисел не более N

```
/**
* @param {number} n
* @return {number}
*/
var countPrimes = function(n) {
};
```

1. Посчитать число простых чисел не более N

```
var countPrimes = function(n) {
    var primes = 0;
    for(i = 2; i < n; i++ ) {
        let isPrime = true;
        for( j=2; j <= i/2; j++) {
            if(i % j == 0) {
                isPrime = false;
                break;
            }
        }
        if(isPrime)
            primes++;
    }
return primes; };

console.log(countPrimes(3));
console.log(countPrimes(10));
speed: O(N^2) memory: O(1)
```

1. Посчитать число простых чисел не более N

```
var countPrimes = function(n) {
    var primes = 0;
    let nonPrimes = {}
    for(i = 2; i <= n/2; i++) {
        if(nonPrimes[i] == undefined) {
            for(j = 2; i*j <= n; j++)
            nonPrimes[i*j] = true;
        }
    }

    for(i = 2; i < n; i++)
        if(nonPrimes[i] == undefined) primes++;
    return primes;
};

console.log(countPrimes(3));
console.log(countPrimes(10));
speed: O(N) memory: O(N)
```

Структура: Hash Table

```
table = {}

table[”key"] = "value2”
```

Вставка значения по ключу:

speed: O(1)

```
table["key1"] != undefined
```

Достать элемент по ключу:

speed: O(1)

2. Вывести дубликаты цифр в массиве

```
let array = [1, 2, 3, 3]
let table = {}

for(i=0; i < array.length; i++) {
    let key = array[i];
    table[key] = table[key] == undefined
        ? 1
        : table[key]++;
        : ++table[key];
}
for(i=0; i < array.length; i++) {
    if(table[array[i]] > 1)
        console.log(array[i])
}
speed: O(N)
memory: O(N)
```

Структура: Set – список уникальных элементов

```
let set = {}

table[”key"] = "value2”
Вставка значения по ключу:
speed: O(1)

table["key1"] != undefined
Проверить есть ли элемент в set-е:
speed: O(1)
```

3. Найти пересечение двух списков (вывести уникальные элементы)

```
/**
* @param {number[]} nums1
* @param {number[]} nums2
* @return {number[]}
*/
var intersection = function(nums1, nums2) {
};
```

Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]
Output: [9,4]

4. Является ли слово анаграммой для другого слова

```
/**
* @param {string} s
* @param {string} t
* @return {boolean}
*/
var isAnagram = function(s, t) {
};
```

Input: *s* = "anagram", *t* = "nagaram"
Output: true

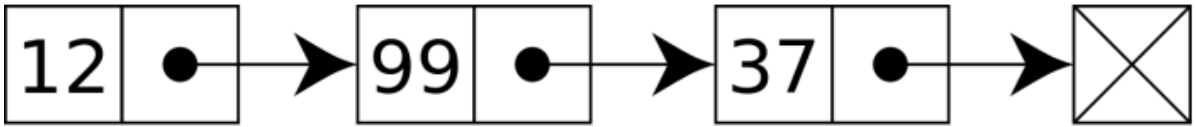
Input: *s* = "rat", *t* = "car"
Output: false

5. Задачи на Linked List

Структура: Linked List

```
Array:          0: 12    1: 99    2: 37

Linked List:    head → 12 → 99 → 37 → ø
```




```
function ListNode(val) {
    this.val = val;
    this.next = null;
}
let head = new ListNode(1);
let tail = new ListNode(2); head.next = tail;

console.log(head)
```

>> ListNode { val: 1, next: ListNode { val: 2, next: null } }

Linked List vs List(Array)

ArrayList:
```
arr = [1, 2, 3]
```

LinkedList:
```
let head = new ListNode(1);
head.next = new ListNode(2);
head.next.next = new ListNode(3);
```

Получить значение по индексу(arr[i]):
ArrrayList: speed: O(1)
LinkedList: speed: O(N)

Добавить элемент в конец списка:
ArrrayList: speed: O(N)
LinkedList: speed: O(1)

1. Развернуть Linked List

Input: 1->2->3->4->5->NULL
Output: 5->4->3->2->1->NULL

```
/**
* function ListNode(val) {
* this.val = val;
* this.next = null;
* }
*/
/**
* @param {ListNode} head
* @return {ListNode}
*/
var reverseList = function(head) {
};
```

1. Развернуть Linked List

```
// Итерирование по LinkedList
var printLinkedList = function(head) {
    let next = head;

    while (next != null) {
        console.log(next.val);
        next = next.next;
    }
};

printLinkedList(head)
```

- 1
- 2
- 3

Ссылки хранятся по значению

```
let head = new ListNode(1);
head.next = new ListNode(2);
head.next.next = new ListNode(3);
// копия ссылки на head
let prev = head;
// копия ссылки на head.next
let curr = head.next;
// копия ссылки на head.next
let next = curr.next;
// по копии ссылки меняем объект
next.next = curr;
// обнулили не объект а ссылку
head = undefined;
// по копии ссылки меняем объект
curr.next = prev;
```

1. Развернуть Linked List

```
var reverseList = function(head) {
    let prev = head;
    let curr = head.next;
    let next = undefined;

    while (curr != null) {
        next = curr.next;
        curr.next = prev;
        prev = curr;
        curr = next;
    }
    return prev;
};
```

>> ListNode { val: 3, next: ListNode { val: 2, next: ListNode { val: 1, next: [Circular] } } }

1. Развернуть Linked List

```
var reverseList = function(head) {
    let prev = head;
    let curr = head.next;
    prev.next = undefined;
    let next = undefined;

    while (curr != null) {
        next = curr.next;
        curr.next = prev;
        prev = curr;
        curr = next;
    }
    return prev;
};
```

1. Развернуть Linked List

```
var reverseList = function(head) {
    if(head == undefined){
        return head;
    }

    let prev = head;
    let curr = head.next;
    prev.next = undefined;
    let next = undefined;
.
.
.
```

2. Определить есть ли цикл в LinkedList

```
function ListNode(val) {
    this.val = val;
    this.next = null;
}
/**
* @param {ListNode} head
* @return {boolean}
*/
var hasCycle = function(head) {
};

// пример linked list с циклом
let head = new ListNode(1);
head.next = new ListNode(2);
head.next.next = head;
```

2. Определить есть ли цикл в LinkedList

```
var hasCycle = function(head) {
    let set = {};
    while(head!=null) {
        if(set[head.val]!=undefined){
            return true;
        }
    set[head.val] = true;
    head = head.next;
    }

    return false;
};
let head = new ListNode(1);
head.next = new ListNode(2);
head.next.next = new ListNode(1);
```

2. Определить есть ли цикл в LinkedList

```
var hasCycle = function(head) {
    let runner1 = head;
    let runner2 = head.next;

    while(runner1!=null && runner2!=null) {
      if(runner1 == runner2) {
       return true;
      }
      runner1 = runner1.next;
      runner2 = runner2.next.next;
    }

    return false;
};
speed: O(N) memory: O(1)
```

2. Определить есть ли цикл в LinkedList

```
var hasCycle = function(head) {
    if(head==undefined){
      return false;
    }

    let runner1 = head;
    let runner2 = head.next;

    while(runner1!=null && runner2!=null && &&
      runner2.next!=null) {
      if(runner1 == runner2) {
       return true;
      }
      runner1 = runner1.next;
    .
    .
```

3. Удалить элемент из списка

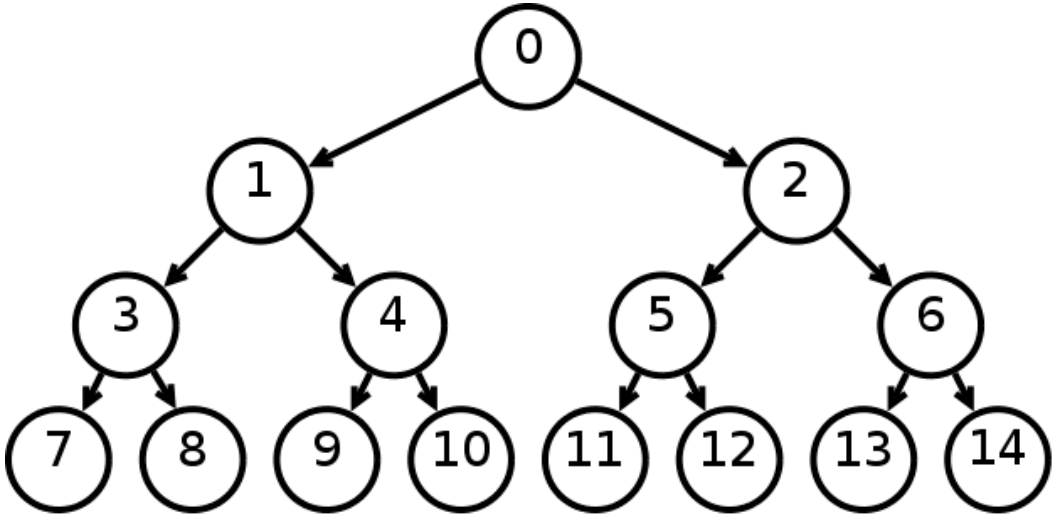
Input: 1->2->6->3->4->5->6, ***val*** = 6

Output: 1->2->3->4->5

```
function ListNode(val) {
this.val = val;
this.next = null;
}
/**
* @param {ListNode} head
* @param {number} val
* @return {ListNode}
*/
var removeElements = function(head, val) {
};
```

6. Деревья

Структура: Бинарное дерево



Дерево у которого у каждого узла не более 2 потомков

Самый известный случай использования – DOM Дерево

Структура: Бинарное дерево

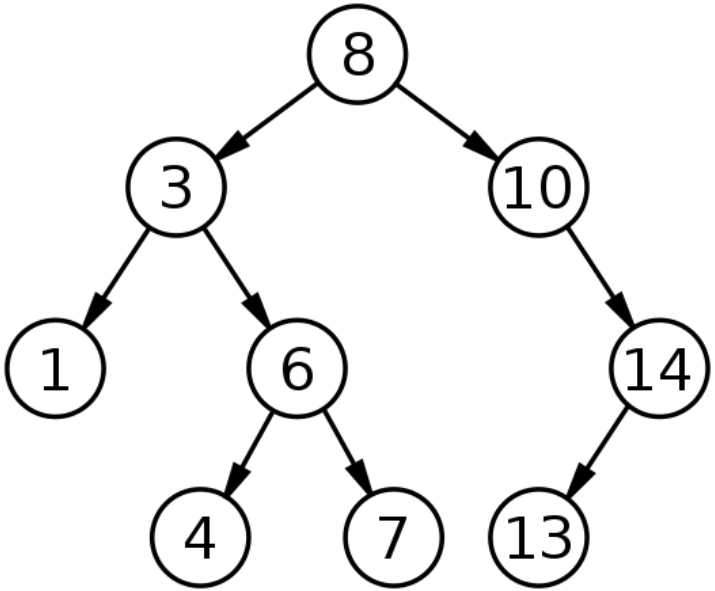
```
function TreeNode(data) {
    this.data = data;
    this.right = null;
    this.left = null;
}

let root = new TreeNode(0);
root.left = new TreeNode(1);
root.right = new TreeNode(1);

console.log(root);
```

>> TreeNode { data: 0,
right: TreeNode { data: 1, right: null, left: null },
left: TreeNode { data: 1, right: null, left: null } }

Структура: Сбалансированное бинарное дерево поиска (Binary Search Tree)



- Оба поддерева — левое и правое — являются двоичными деревьями поиска.
- У всех узлов *левого* поддерева произвольного узла X значения ключей данных *меньше*, нежели значение ключа данных самого узла X.
- У всех узлов *правого* поддерева произвольного узла X значения ключей данных *больше либо равны*, нежели значение ключа данных самого узла X.

Структура: Бинарное дерево поиска

Алгоритмическая сложность:

Поиск:

```
speed: O(logN)
```

Удаление:

```
speed: O(logN)
```

Вставка:

```
speed: O(logN)
```

Структура: Бинарное дерево поиска

```
class TreeNode {

    constructor(data) {
        this.data = data;
        this.right = null;
        this.left = null;
    }

    add(data) {}

    find(data) {}

    all() {}
}
```

Бинарное дерево поиска: find

```
let root = new TreeNode(5);

let left = new TreeNode(4);
let right = new TreeNode(6);

root.left = left;
root.right = right;

console.log(root.find(3))
console.log(root.find(5))
console.log(root.find(6))
```

Бинарное дерево поиска: find

```
find(data) {
    let next = this;
    while(next!=null) {
        if(next.data > data) {
            next = next.left;
        } else if (next.data < data){
            next = next.right;
        } else {
            return next;
        }
    }

    return null;
}
```

Бинарное дерево поиска: add

```
let root = new TreeNode(7);

root.add(3);
root.add(10);
root.add(4);
root.add(5);

console.log(root)
```

Бинарное дерево поиска: add

```
add(data) {
    let next = this;
    while(true) {
        if(next.data === data) {
          break;
        } else if(next.data < data) {
            if(next.right == null){
              next.right = new TreeNode(data);
              break;
            } else{
              next = next.right;
            }
        } else {
            if(next.left == null){
              next.left = new TreeNode(data);
              break;
            } else{
              next = next.left;
            }}}}
```

Бинарное дерево поиска: all
Обход в глубину (DFS Deep First Search)

```
dfs(all, node) {
    if(node!=null) {
      all.push(node.data);
      this.dfs(all, node.left);
      this.dfs(all, node.right);
      }
    }

    all() {
      let all = [];
      this.dfs(all, this);
      return all;
    }
}
```

Проверить является ли дерево сбалансированным

```
/**
* Definition for a binary tree node.
* function TreeNode(val) {
* this.val = val;
* this.left = this.right = null;
* }
*/
/**
* @param {TreeNode} root
* @return {boolean}
*/
var isBalanced = function(root) {
};
```