**笔试**

1. **合并二叉树**

给定两棵二叉树，这两棵树的部分节点是重叠的，部分没重叠。

现在要将它们合并到一个新的二叉树中。合并规则是，如果两个节点重叠，则将两个节点的值相加，作为合并节点的新值；否则，非空节点的值作为新树的节点。

例如：

|  |
| --- |
| **Input:**  Tree 1 Tree 2  1 2  / \ / \  3 2 1 3  / \ \  5 4 7  **Output:**  Merged tree:  3  / \  4 5  / \ \  5 4 7 |

/\*\*

\* 已有函数：生成一个二叉树

\* function TreeNode(val) {

\* this.val = val;

\* this.left = this.right = null;

\* }

\*/

/\*\*

\* @param {TreeNode} t1

\* @param {TreeNode} t2

\* @return {TreeNode}

\*/

var mergeTrees = function(t1, t2) {

// 填入代码

    var t;

    if (t1 !== null && t2 !== null) {

        t.val =  t1.val + t2.val;

    } else if (t1 !== null && t2 === null) {

        t = t1

    } else if (t1 === null && t2 !== null) {

        t = t2

    } else {

        return null;

    }

    return t;

};

1. **子串问题**

给定两个字符串s1和s2，如果s2包含s1的任一排列，则返回true。

换句话说，第一个字符串的排列之一，是第二个字符串的子串。

|  |
| --- |
| **例1：**  Input: s1 = "ab", s2 = "eidbaooo"  Output: True  说明: s2包含s1的一个排列("ba")  **例2：**  Input: s1= "ab", s2 = "eidboaoo"  Output: False |

注：

1）输入的字符串只包含小写字母。

2）两个字符串的长度在[1, 10,000]范围内。

/\*\*

\* @param {string} s1

\* @param {string} s2

\* @return {boolean}

\*/

var checkInclusion = function(s1, s2) {

// 填入代码

    var a1=s1.split("");

    var ra1=a1.reverse();

    var jra1=ra1.join("");

    if(s2.indexOf(s1)>0 || s2.lastIndexOf(rja1)>0){

        return true;

    }else{

        return false;

};

1. **奇偶链表**

给定一个单向链表，将所有奇数节点组合在一起，然后是偶数节点。需要注意的是，这里指的是节点序号而不是节点中的值。

例：

输入1-> 2-> 3-> 4-> 5-> NULL，

输出1-> 3-> 5-> 2-> 4-> NULL。

注意：

1）偶数和奇数组内的相对顺序应与输入时保持相同。

2）第1个节点是奇数，第2个节点是偶数，以此类推。

/\*\*

\* Definition for singly-linked list.

\* function ListNode(val) {

\* this.val = val;

\* this.next = null;

\* }

\*/

/\*\*

\* @param {ListNode} head

\* @return {ListNode}

\*/

var oddEvenList = function(head) {

// 填入代码

    if (!head ||!head.next) {

        return head;

    }

    var frist=head,

        second=head.next,

        third=head.next.next;

    var fristHead = first,

        secondHead = second;

    var Odd = true;

    while (third !== null) {

        if (Odd) {

            frist.next = third;

            frist = third;

        } else {

            second.next = third;

            second = third;

        }

        third = third.next

        Odd = !Odd

    }

    if (!Odd) {

        second.next = null;

    }

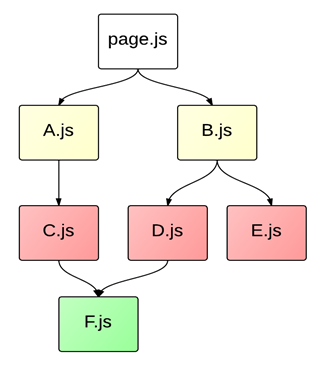
    frist.next = secondHead;

    return fristHead;

};

1. **Combo url**

页面page的主函数入口在page.js，以下为其依赖树。



为了性能要求，我们页面会组装一个combo请求：

http://res.wx.qq.com/F.js,E.js,D.js,C.js,B.js,A.js,page.js

请设计genUrl(requireTree)，输出如上所示的combo url。

requireTree的数据结构如下页。

requireTree = {

"name" : "page.js"

"require" : [

{

"name":"A.js"

"require" : [

{

"name":"C.js",

"require" : [

{

"name":"F.js"

}

]

}

]

},

{

"name":"B.js"

"require" : [

{

"name":"D.js",

"require" : [

{

"name":"F.js"

}

]

},

{

"name":"E.js",

"require" : []

}

]

},

]

}

/\*\*

\* @param {object} requireTree

\* @return {string}

\*/

var genUrl = function(requireTree) {

// 填入代码

};