伏城之外 ⑤ 已于 2023-03-11 16:14:32 修改 ● 7029 🏚 收藏 14 版权 分类专栏: 华为OD机试 (Java & DD, JS & Samp; Python) # 华为OD机试2023A 文章标签: 算法 华为机试 Python JavaScript Java 华为OD机试2... 同时被 2 个专栏收录▼ ¥49.90 OD 已订阅 253 订阅 132 篇文章 ለ 该专栏为热销专栏榜 第46名 ¥99.00 题目描述 求 <mark>单向链表^Q 中间的节点值,如果奇数个节点取中间,偶数个取偏右边的那个值。</mark> 输入描述 第一行 链表头节点地址 后续输入的节点数n 后续输入每行表示一个节点,格式 节点地址 节点值 下一个节点地址(-1表示 空指针^Q) 输入保证链表不会出现环,并且可能存在一些节点不属于链表。 输出描述 单向链表中间的节点值 用例 000104 000003-1 输入 00010 5 12309 11451 6 00000 12309 7 11451 输出 6 说明 无 100003 76892 7 12309 输入 12309 5 -1 10000 1 76892 输出 7 说明 无 题目解析 用例1示意图如下 cur val nxt val nxt val nxt val nxt cur cur cur 12309 00010 5 12309 7 11451 11451 6 00000 00000 3 -1 ❷SDN @伏城之外 JS本题可以利用数组模拟链表 基于链表数据结构解题 JavaScript算法源码 const readline = require("readline"); 2 3 4 const rl = readline.createInterface({ input: process.stdin, output: process.stdout, }); 8 const lines = []; 9 let head; 10 let n; 11 rl.on("line", (line) => { 12 lines.push(line); 13 14 if (lines.length === 1) { 15 [head, n] = lines[0].split(" "); 16 } 17 18 if (n && lines.length === n - 0 + 1) { 19 lines.shift(); 20 21 22 const nodes = {}; 23 lines.forEach((line) => { 24 const [addr, val, nextAddr] = line.split(" "); 25 nodes[addr] = [val, nextAddr]; 26 }); 27 28 console.log(getResult(head, nodes)); 29 30 lines.length = 0; 31 32 }); 33 34 function getResult(head, nodes) { 35 const linkedlist = []; 36 37 let node = nodes[head]; 38 while (node) { 39 const [val, next] = node; 40 41 linkedlist.push(val); 42 node = nodes[next]; 43 } 44 45 const len = linkedlist.length; 46 47 const mid = len % 2 === 0 ? len / 2 : Math.floor(len / 2); 48 49 50 return linkedlist[mid]; 51 | } Java算法源码 需要注意的是Java中LinkedList类的get(index)方法的时间复杂度不是O(1),而是O(n),这题建议使 用ArrayList代替 import java.util.ArrayList; 1 import java.util.HashMap; 2 import java.util.Scanner; 3 4 public class Main { 5 public static void main(String[] args) { 6 Scanner sc = new Scanner(System.in); 8 9 String head = sc.next(); 10 int n = sc.nextInt(); 11 12 HashMap<String, String[]> nodes = new HashMap<>(); for (int i = 0; i < n; i++) { 13 String addr = sc.next(); 14 15 String val = sc.next(); String nextAddr = sc.next(); 16 nodes.put(addr, new String[] {val, nextAddr}); 17 18 19 System.out.println(getResult(head, nodes)); 20 } 21 22 public static String getResult(String head, HashMap<String, String[]> nodes) 23 24 ArrayList<String> link = new ArrayList<>(); 25 26 27 String[] node = nodes.get(head); 28 while (node != null) { String val = node[0]; 29 String next = node[1]; 30 31 link.add(val); 32 node = nodes.get(next); 33 34 35 36 int len = link.size(); int mid = len / 2; 37 return link.get(mid); 38 39 40 Python算法源码 head, n = input().split() 2 3 4 $nodes = \{\}$ for i in range(int(n)): 5 addr, val, nextAddr = input().split() 6 nodes[addr] = [val, nextAddr] 7 8 9 10 def getResult(head, nodes): 11 linkedlist = [] 12 node = nodes.get(head) 13 14 15 while node is not None: 16 val, next = node 17 linkedlist.append(val) node = nodes.get(next) 18 19 length = len(linkedlist) 20 mid = int(length / 2) 21 22 23 return linkedlist[mid] 24 25 26 # 算法调用 27 print(getResult(head, nodes)) 快慢指针解题 链表数据结构本质上来说没有索引概念,因为其在内存上不是一段连续的内存,因此索引对于链表 结构而言没有意义。 但是从使用上来说,我又经常需要去获取链表结构的第几个元素,因此大部分语言都为链表结构提 高了"假索引",比如Java的LinkedList类,虽然提高了get(index)方法,但是其底层是通过遍历链 表^〇(通过next属性找到下一个节点)来找到对应"假索引"的元素的,即LinkedList每次都需要O(n) 的时间复杂度才能找到index位置上的元素。 另外,链表还有一个常考问题,那就是链表长度未知的情况下,我们如何找到链表的中间节点? 此时,就要用到快慢指针。 所谓快慢指针,即通过两个指针遍历链表,慢指针每次步进1个节点,快指针每次步进2个节点,这 样快指针必然先到达链表尾部,而当快指针到达链表尾部时,慢指针其实刚好就是在链表中间节点 的位置(奇数个节点取中间,偶数个取偏右边的那个值)。 本题虽然给出了节点数,但是这些节点不一定属于同一个链表结构,因此本题的链表长度也是未知 的,而本题要求的链表中间节点要求刚好和快慢指针找的中间节点吻合,因此本题最佳策略是使用 快慢指针。 Java算法源码 import java.util.HashMap; 2 import java.util.Scanner; 3 4 public class Main { public static void main(String[] args) { Scanner sc = new Scanner(System.in); 6 7 String head = sc.next(); 8 int n = sc.nextInt(); HashMap<String, String[]> nodes = new HashMap<>(); 11 for (int i = 0; i < n; i++) { 12 String addr = sc.next(); 13 String val = sc.next(); 14 String nextAddr = sc.next(); 15

Q 搜索

♦ 单向链表中间节点

SDN 博客

n)

下载

学习

社区 GitCode InsCode

华为OD机试 – 单向链表中间节点(Java & JS & Pytho

28 29 30 31 32

33

34

35

36

37

38

39

1

2

3

4

5

6

7

0

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

算法调用

nodes[addr] = [val, nextAddr]

def getResult(head, nodes):

slow = nodes.get(head)

while fast is not None:

if fast is None:

break

print(getResult(head, nodes))

else:

return slow[0]

fast = nodes.get(slow[1])

slow = nodes.get(slow[1])

fast = nodes.get(fast[1])

fast = nodes.get(fast[1])

});

16

17

18

19

20

21

22

23

24

25

26

27

}

nodes.put(addr, new String[] {val, nextAddr});

public static String getResult(String head, HashMap<String, String[]> nodes)

System.out.println(getResult(head, nodes));

String[] slow = nodes.get(head);

slow = nodes.get(slow[1]);

fast = nodes.get(fast[1]);

const readline = require("readline");

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout,

fast = nodes.get(fast[1]);

while (fast != null) {

if (fast != null) {

} else {

}

JavaScript算法源码

break;

return slow[0];

String[] fast = nodes.get(slow[1]);

9 const lines = []; let head; 10 let n; 11 rl.on("line", (line) => { 12 lines.push(line); 13 14 15 if (lines.length === 1) { [head, n] = lines[0].split(" "); 16 17 } 18 19 if (n && lines.length === n - 0 + 1) { lines.shift(); 20 21 const nodes = {}; 22 23 lines.forEach((line) => { 24 25 const [addr, val, nextAddr] = line.split(" "); nodes[addr] = [val, nextAddr]; 26 }); 27 28 console.log(getResult(head, nodes)); 29 30 lines.length = 0; 31 32 }); 33 34 function getResult(head, nodes) { 35 let slow = nodes[head]; 36 37 let fast = nodes[slow[1]]; 38 while (fast) { 39 slow = nodes[slow[1]]; 40 41 fast = nodes[fast[1]]; 42 43 if (fast) { fast = nodes[fast[1]]; 44 45 } else { 46 break; 47 48 49 return slow[0]; 50 51 Python算法源码 2 head, n = input().split() 3 $nodes = \{\}$ 4 for i in range(int(n)): addr, val, nextAddr = input().split() 6